Department of Veterans Affairs

Office of Information and Technology (OIT) Product Development (PD)

VA Enterprise Target Application Architecture: SOA Layer Implementation Guide

DRAFT

V 0.1

January 9, 2012

Document Control

Amendment Record

Issue	Date	Author	Comments
0.1 (Draft)	1/9/2012	Jeffrey Mohr	Initial Draft

The latest version of this document supersedes all other previous issues.

Distribution Record

Issue	Date	Recipients
0.1	1/9/2012	ASD and PD Management and Technical Review

Index of Acronyms & Abbreviations

Acronym	Definition
AAA	Authentication, Authorization and Auditing
API	Application Programming Interface
BPEL	Business Process Execution Language
BPM	Business Process Management
BRM	Business Rules Management
C&A	Certification and Accreditation
CAIP	CBP Application Integration Project
CDP	Common Delivery Platform
CRUD	Create Read Update Delete
DAO	Data Access Object
DB	Database
EAA	Enterprise Target Application Architecture
EJB	Enterprise Java Beans
ESB	Enterprise Service Bus
ESSO	Enterprise Single Sign-On
ETA	Enterprise Technical Architecture
HA	High Availability
HTTP	Hypertext Transfer Protocol
IRI	Internationalized Resource Identifier
Java EE	Java Platform, Enterprise Edition
JBI	Java Business Integration
JCA	Java Connector Architecture
JDBC	Java Database Connectivity
JMS	Java Message Service
JPA	Java Persistence API
JSON	JavaScript Object Notation
JSR	Java Specification Request
JVM	Java Virtual Machine
MQ	Message Queue
ORM	Obejct Relational Mapping
PII	Personally Identifiable Information
POJO	Plain Old Java Object
PTA	Privacy Threshold Assessment
PUMA	Privileged User Management Administration
QoS	Quality of Service
REST	Representational State Transfer

RIA	Rich Internet Application
RMI	Remote Method Invocation
SIEM	Security Information and Event Management
SOA	Service Oriented Architecture
SOAP	Simple Object Access Protocol
SQL	Structured Query Language
SSO	Single Sign-On
UI	User Interface
USGCB	United States Government Configuration Baseline
WSDL	Web Service Definition Language
WSIF	Web Services Invocation Framework
XML	Extensible Markup Language
XSLT	Extensible Style sheet Language Transformations

Contents

Tables

Figures

1. Introduction

The long term architectural direction posited in the VA Enterprise Target Application Architecture (EAA) calls for presentation services to be implemented as portlets in a portal and business logic services to be implemented as SOA services in an Enterprise Service Bus (ESB). Further, the security model for the SOA services is based on the ability to verify identities across the enterprise, a capability to be provided by an enterprise single sign-on deployment. However, at the present time neither the portals nor the ESB are implemented and deployed and thus, are not available for use by service developers. Further, the single sign-on solution has not yet been selected or deployed. Because these components of the architecture are not available for developers, the initial presentation, business logic, and data logic will need to be implemented using standard object oriented development approaches. This document describes the development of presentation, business logic, and data logic services.

Further, while the VA EAA is based on the use of VA standard messages for communications between services belonging to separate applications, it does not require the use of such messages for services within an application. This document describes how presentation, business, and data services will be built and will communicate within an application.

1.1. Background

The EAA is defined in terms of a series of layers, each layer providing specific capabilities needed to support the deployment and operation of VA systems. The top most layer is the SOA Services layer which includes sub-layers for the presentation services, business services, and data services that access the Data Layer, a much lower layer in the hierarchy that supports access to the physical data.

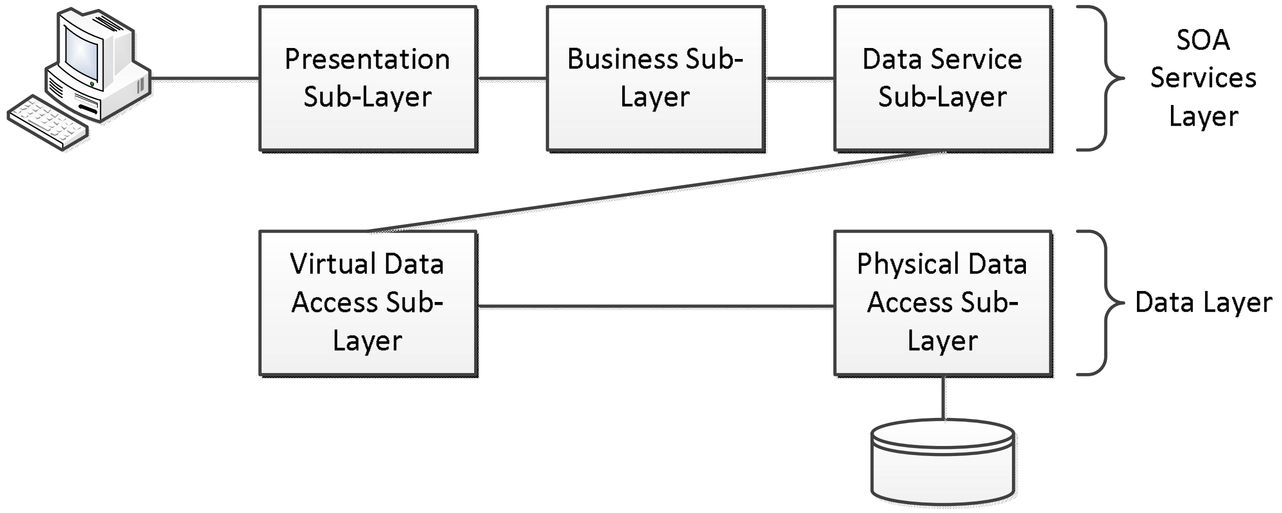


Figure 1 SOA and Data Layers of the EAA

Applications will be composed of a series of presentation, business, and data services. The data services exist at two levels in the architecture – the SOA layer where the data services are primarily pass through services and the Data Layer in which the physical access to the data is performed and the data is packaged for use by the business services.

1.2. Purpose

While initial implementations of services in conformance with the EAA and the SOA Technical Framework will be based on the use of standard object oriented programming techniques because the portals and the ESB will not yet be available, there is a need for guidance how to implement those services. This document is intended to provide guidance as to how to develop presentation, business, and data services within an application directly and to allow them to communicate without the need to convert data to the VA standard messages.

1.3. Audience

This document is primarily written for VA solution architects, developers and engineers who will be architecting, designing and developing the VA Enterprise Applications and the related infrastructure. This document is also pertinent to other application/system stakeholders such as members of the Architecture and Engineering Review Board (AERB), technology strategy, information technology transformation, and Architectural Assessment & Alignment (AAA) teams. This document is also intended to benefit any VA technologist who is associated with VA system engineering and system development lifecycles.

1.4. Document Conventions

The specific architecture guidelines described in this document fall into two categories:

Mandatory Compliance: These guidelines are identified by the key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT". Exceptions require a waiver and a transition plan.

Recommended use: These guidelines are identified by the key words "SHOULD", “RECOMMENDED”, "SHOULD NOT", “NOT RECOMMENDED”. These guidelines describe a preferred alternative. Deviation should only be on a limited use and justified by the circumstances.

2. Presentation Sub-Layer

2.1. Presentation Sub-Layer Architecture

The VA Presentation Sub-Layer architecture will be based on a loosely coupled model with a clear separation of presentation responsibilities such as User Interface elements and Presentation Logic from the Business logic. Enforcing a clean separation between the business and presentation logic greatly enhances code reuse and testability and allows the services at either level to be changed without requiring testing or integration with the other – as long as the external characteristics of the services are maintained. The Presentation Sub-Layer architecture will depend on the application archetypes listed below.

- **Dynamic Web Applications** – “Thin” user interfaces
    - Traditional HTML/XHTML based apps
    - AJAX based Rich Client components
- Sandbox/Plug-in based Rich Internet Applications
- Mobile Applications

The key architectural considerations for the Presentation Sub-Layer are that it;

- MUST contain Presentation functionality only. It must not mix Business or Data Access logic in presentation components.
- MUST be loosely coupled with the Business Logic.

- MUST interface with Business Logic Sub-Layer for business logic processing and the related data logic. The Presentation Sub-Layer MUST NOT persist any business data to the database of record directly.
- Implementation of standalone thick clients, on the client tier is NOT PERMITTED. However in specific scenarios where device integration is needed and where an applet functionality will not suffice, because of the stringent requirements and the complexity involved, a thick client MAY BE considered in the architecture and used only by specific EAA waiver.
- All VA applications SHALL provide a common look and feel to the users.
- Information MUST be accessible to all users, in accordance with Section 508 of the Rehabilitation Act of 1998, as amended, 29 USC 794(d).
- When persistent cookies are used, they MUST NOT contain sensitive, or Personally Identifiable Information (PII) or Protected Health Information (PHI).

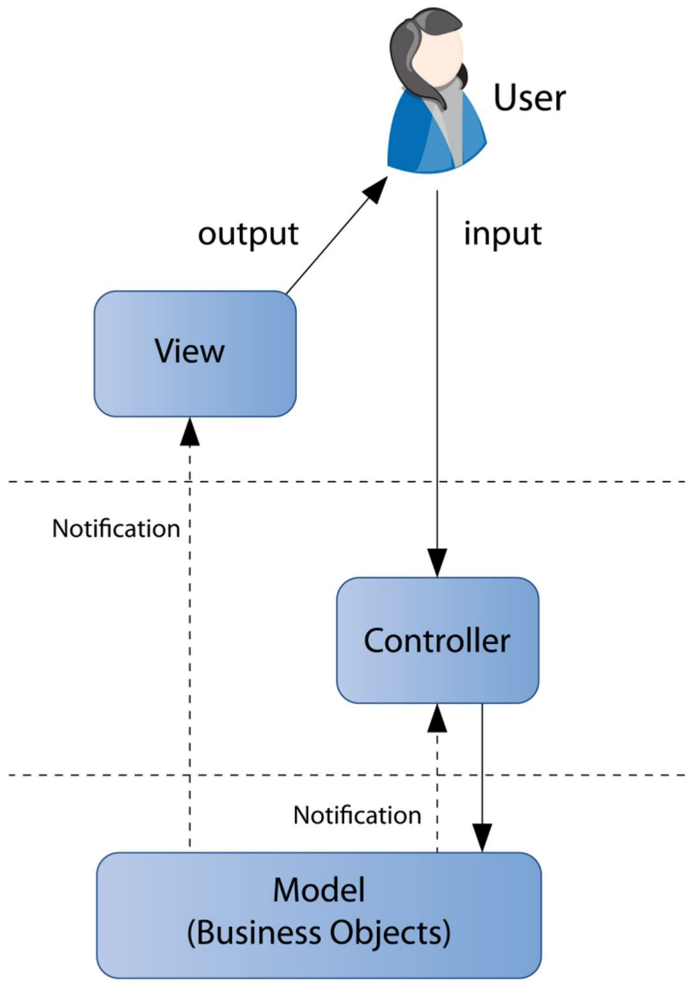


Figure 2 - MVC Pattern

- All VA browser based applications SHALL provide cross browser support across VA approved web browsers.
- The Presentation component design SHOULD follow a test driven approach.

2.2. Architectural Patterns

The Presentation Sub-Layer design SHOULD follow the Model-View-Controller (MVC) design pattern or a variant such as the Model View Presenter (MVP) design pattern for all application archetypes.

2.2.1. Model View Controller (MVC) Pattern

The MVC pattern separates the business data from the presentation and control logic that uses this functionality, allowing multiple views to share the same data model, resulting in presentation code that is easy to maintain. In an MVC pattern,

- The model represents enterprise data and the business rules that govern access to and updates of this data.
- The view renders the contents of the model. It is the view's responsibility to maintain consistency in its presentation when the model changes. This can be achieved by using a push model, where the view registers itself with the model for change notifications, or a pull model, where the view is responsible

for calling the model when it needs to retrieve the most current data.

- The controller translates interactions with the view into actions to be performed by the model. In the case of RIA, the controller will typically reside on the client; while in the dynamic web architecture, the controller is typically implemented as part of the presentation logic on the server side.

Shown below is a variation of MVC called MVC2 pattern that incorporates a clear separation of view and controller. A controller receives all the requests, retrieves data from a Model and forwards it to next view for presentation.

2.2.2. **Model View Presenter (MVP) Pattern**

The MVP pattern is a derivative of the MVC pattern. In the MVP Pattern the View is more loosely coupled to the Model and the presenter is responsible for binding the model to the view. The Presenter's purpose is to interpret events and perform the logic necessary to map them to the proper commands to manipulate the model in the intended fashion. Most of the user interface code is included in the Presenter. Basically, in MVP there is no Application Model middle-man since the Presenter assumes this functionality. Additionally, the View in MVP is responsible for handling the UI events, which used to be the Controllers job, and the Model becomes strictly a Domain Model.

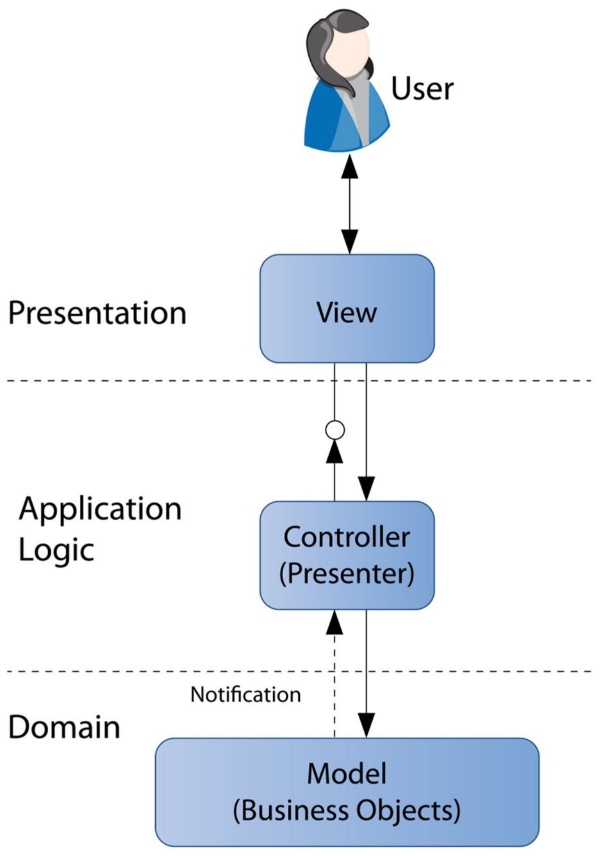


Figure 3 - MVP Pattern

2.3. Design Standards

2.3.1. **Web Design Standards**

The web design standards outlined in this section are based on W3C, VA and usability.gov web standards.

2.3.1.1. **Markup**

Structural markup should be used to promote consistency in documents and supply information to other tools (e.g., indexing tools, search engines, programs that extract tables to databases, navigation tools that use heading elements, and automatic translation software that translates text from one language into another). The best practices for markup are:

- Markup tags should follow the W3C specification. The tags should be well formed (with proper closing tags) to ensure consistent rendering across browsers.

- Markup should be consistently formatted to make it readable, portable and reusable by other developers.

- Markup should follow the industry standard practice of being lower-cased, as the case affects code readability.

- The code should be written to make dynamic content more accessible or provide an alternative presentation/page.

- The primary language should be identified with markup.
- Markup abbreviations and acronyms should be used to indicate the expansion.
- Quotation markup should not be used for formatting effects such as indentation.
- Meaningful Title (<title>) and Meta Tags should be used to make the application web pages more meaningful and search-engine friendly.
- Metadata should be provided to add semantic information to pages and sites. For each web page, a concise and relevant summary should be placed inside the META description tag.
- It is a good practice to have precise and shorter comments as a part of Mark code. However avoid describing the business logic as a part of the comments.
- Ensure that foreground and background color combinations provide sufficient contrast when viewed by someone having color deficits or when viewed on a black and white screen.

2.3.1.2. **Cascading Style Sheets**

Cascading Style Sheets (CSS) provides a greater control on the layout and look and feel of the content in the page. Using style sheet leads to the standardization of look and feel of the content across the enterprise. The best practices for CSS are:

- An external stylesheet should be used instead of embedded style sheets to separate content from presentation. External stylesheets can be cached allowing faster web page loading times.
- Inline styles should be avoided in CSS because it mixes up content and presentation.
- The CSS should use a reset stylesheet on the top in order to start out by setting every element to have certain styles.
- The CSS code should be tested/validated and then minified[1] by removing all the extra whitespace, such as tabs, spaces and newlines. This reduces the CSS file size, helping to speed up page loading times.
- A color reference should be made at the top of the CSS file. This makes it easier to find the code for a specific color.
- CSS should be used for formatting instead of deprecated HTML tags and attributes as it will result in cleaner, easier-to-maintain code that downloads faster.
- CSS should be used for page layout instead of tables for cleaner, easier-to-maintain code that downloads faster and it will make the web pages more accessible. CSS gives more control over the look and feel of web pages.
- CSS shorthand should be used to speed up the code development time and make the web pages download faster.

[1] Minification in computer programming languages and especially JavaScript, is the process of removing all unnecessary characters from source code, without changing its functionality. These unnecessary characters usually include white space characters, new line characters, comments, and sometimes block delimiters, which are used to add readability to the code but are not required for it to execute.

2.3.1.3. **Scripting**

The recommended best practices for using scripting in web applications are:

- JavaScript-enabled buttons should not be used to refresh the contents of the screen. JavaScript controls (for example, rollover controls) shall be perceivable via assistive technology and operable via the keyboard.
- Creation of links that use "JavaScript" as the URI should be avoided. If a user is not using scripts, then they won't be able to link since the browser cannot create the link content.
- Declaration of a global of a function-level variable should be prefixed with “var” keyword.
- Scripts code should be written to use feature-detection instead of browser-detection. That is, before using any advanced feature that an older browser may not support, the function or property should be checked for existence first, and then may be used if available.
- Square Bracket Notation should be used when accessing object properties that are determined at run-time or which contain characters not compatible with dot notation.
- The eval() function in JavaScript should not be used as it is a way to run arbitrary code at run-time. eval() decreases the script’s performance substantially and also poses a huge security risk because it grants far too much power to the provided text.
- The forms and form elements should be referenced correctly. All forms in an HTML form should have a name attribute. For XHTML documents, the name attribute is not required and instead the form tag should have an id attribute and should be referenced using document.getElementById(). Referencing forms using indexes, such as document.forms[0] is a bad practice in almost all cases. Some browsers make the form available as a property of the document itself using its name. This is not reliable and shouldn't be used.
- The script blocks should not use HTML comments as no browsers in common use today are ignorant of the <script> tag, so hiding of JavaScript source is no longer necessary.
- A single global namespace should be used to encapsulate functionality. The global namespace should not be cluttered: Using global namespaces may cause naming conflicts between JavaScript source files and cause code to break.
- Sync "Ajax" calls should be avoided. Sync mode will cause the browser to lock up for the user and wait for the request to return before continuing.
- JavaScript Object Notation (JSON) should be used instead of XML when storing data structures as plain text or sending/retrieving data structures via Ajax, JSON is a more compact and efficient data format, and is language-neutral.
- Place scripts at the bottom of the web page.
- JavaScript events should never be included as inline attributes.
- All JavaScript behaviors should be included in external script files and linked to the document with a <script> tag in the head of the page.
- The use of image sprites is recommended for images on frequently loaded pages to decrease numbers of requests.

2.3.1.4. **Graphics**

Below are the recommended best practices for using graphics in the web applications:

- Graphics should be created by working in RGB mode and web-safe colors should be used.
- The image quality should be balanced with the file size. Create images with the precise size needed.
- The graphics files should be optimized for the web. Small graphics file sizes should be created for distribution of images on the web. With smaller file sizes, web servers can store and transmit images more efficiently, and viewers can download images more quickly.
- The best format should be chosen for the web graphics. Different types of graphics need to be saved in different file formats to optimize display and create a file size suitable for the web.
- Reduce the number of colors when possible.
- Images should be compressed and optimized to decrease load time and make a site more accessible.
- Compression should be set for the best compromise between file size & image quality.
- Graphics should not be used as text unless absolutely necessary.
- Images should not be resized in HTML. Image software should be used if resizing is necessary.
- Portable Network Graphics (PNG), a bitmapped image format that employs lossless data compression, should be used to support illustrations and photographs. PNG 8 format efficiently compresses solid areas of color while preserving sharp detail and supports transparent background
- Scalable Vector Graphics (SVG) format should be used for interactive line art, data visualization, and even user interfaces need the power of SVG.

2.3.1.5. **Multimedia**

The recommended best practices for using multimedia features in web applications are:

- A text equivalent for every non-text element (e.g., via "alt", "longdesc", or in element content) should be provided.
- When necessary, a text equivalent should be provided for visual information to enable understanding of the page.
- The audio/video/Flash files should be used to enhance the web page in the application rather than distract from the site. Each audio/video/Flash file used should serve a clear purpose.
- Captions should be provided for each audio or video file used (accessibility).
- Timed Text should be used for the purpose of interchange among authoring systems. It may also be used directly as a distribution format, thus suitable for captioning or video description.
- Download times for audio or video files should be indicated.
- Links to download for media plug-ins should be provided.
- Media annotations should be used to provide ways to describe media resources, using a common set of properties. Media annotations help convey information that can then be reused in search engines or tagging systems
- Media fragments should be used to provide a media-format independent on the Web using identifiers (URL, URI, IRI).

2.3.1.6. **Accessibility**

Section 508 Amendment to the Rehabilitation Act of 1973 (Section 508) was enacted to make Federal systems accessible to users with disabilities. Section 508 requires Federal agencies to ensure that their procurement, development, maintenance, or use of electronic and information technology offers all end users equivalent access to that technology, except in the event of exceptions or undue burdens as specified in the Section 508 legislation. While Section 508 compliance is mandatory for all Federal agencies, it is of particular importance for VA systems because of the substantial number of Service Disabled Veterans in the VA user population.

Section 508 requires that federal agencies' electronic and information technology is accessible to people with disabilities, including employees and members of the public. The types of disabilities and associated challenges related to Section 508 include mobility/motor skills impairment (dexterity), vision impairments, hearing impairments, and speech impairments. Assistive technology is adaptive equipment that people with disabilities commonly used for information and communication access. The Section 508 standards can be found at www.section508.gov.

Software, including Web sites, must also be designed to ensure complete compliance with Section 508. The Section 508 Compliant option provides the following:

- Images, dynamic elements, graphics/media, forms, should include "Alt" text (alternative text). This text allows visually impaired people using a screen reader (or assistive software, such as JAWS) to hear or read via Braille where they are in the web content output and also informs them as to the buttons that they are pushing.
- Navigation frames should have "names" and "titles". The title is readable by assistive software.
- Headings and labels should be descriptive.
- The purpose of a link on the web page should use the link text combined with the text of the enclosing sentence.
- HTML tables should be generated in a way so that rows and column headers can be identified by screen readers or assistive technology.
- If images, Flash, or DHTML is the main navigation, clear text links should be present in the footer section of the page.
- Navigation aids such as site map, glossary, table of contents, breadcrumbs and help link on every page should be provided.
- Color should not be used alone to convey meaning. The information conveyed by color differences should also be available in text.
- Captions should be provided for each audio or video file used. Include a sign language interpreter in the video stream.
- If the web site uses frames, frame titles should be used and meaningful content should be placed in the "no frames" area.
- To assist screen readers, the html element's lang and xml:lang attributes should be configured to indicate the spoken language of the page.

2.3.2. **USGCB Compliance**

United States Government Configuration Baseline (USGCB) administered by National Institute for Standards and Technologies (NIST) and adopted by VA, succeeds the Federal Desktop Core Configuration

(FDCC) initiative for Desktop Management. The web applications should consider special operational restrictions imposed by USGCB while designing the presentation components.

- USGCB blocks end users from installing ActiveX controls. If the presentation components require ActiveX controls, they should be distributed using the central Desktop Management support.
- USGCB also requires Java runtime on client machines be used in the Intranet and Trusted Sites zones only. Applet or Thick client component development should consider deployment for trusted zones only.

2.3.3. **Page Layouts**

This section outlines the Wire frames, and high level page layouts. The high-level design guidelines for the VA Web pages are:

- Pages should appeal to the target audience
- Use consistent site headers/logos
- Use a consistent navigation area
- Use informative page titles that includes the organization/site name
- Page footer areas should include — copyright, last update, contact e-mail address
- Good use of basic design principles include: repetition, contrast, proximity, and alignment
- Displays should be designed to not require horizontal scrolling at 1024x768 and higher resolutions
- Balance of text/graphics/white space on page
- Provide good contrast between text and background
- Ensure that repetitive information (header/logo and navigation) takes up no more than one-quarter to one-third of the top portion of the browser window at 1024x768 resolution
- Home page has compelling, interesting information above the fold (before scrolling down) at 1024x768
- Home page downloads within 10 seconds on dial-up connection

2.3.3.1. **Wire Frames**

A website wireframe, also known as a page schematic or screen blueprint, is a visual guide that represents the skeletal framework of a website. The wireframe depicts the page layout or arrangement of the website's content, including interface elements and navigational systems, and how they work together. The wireframes provided below display the standard look and feel, navigation, and color schemes for VA applications.

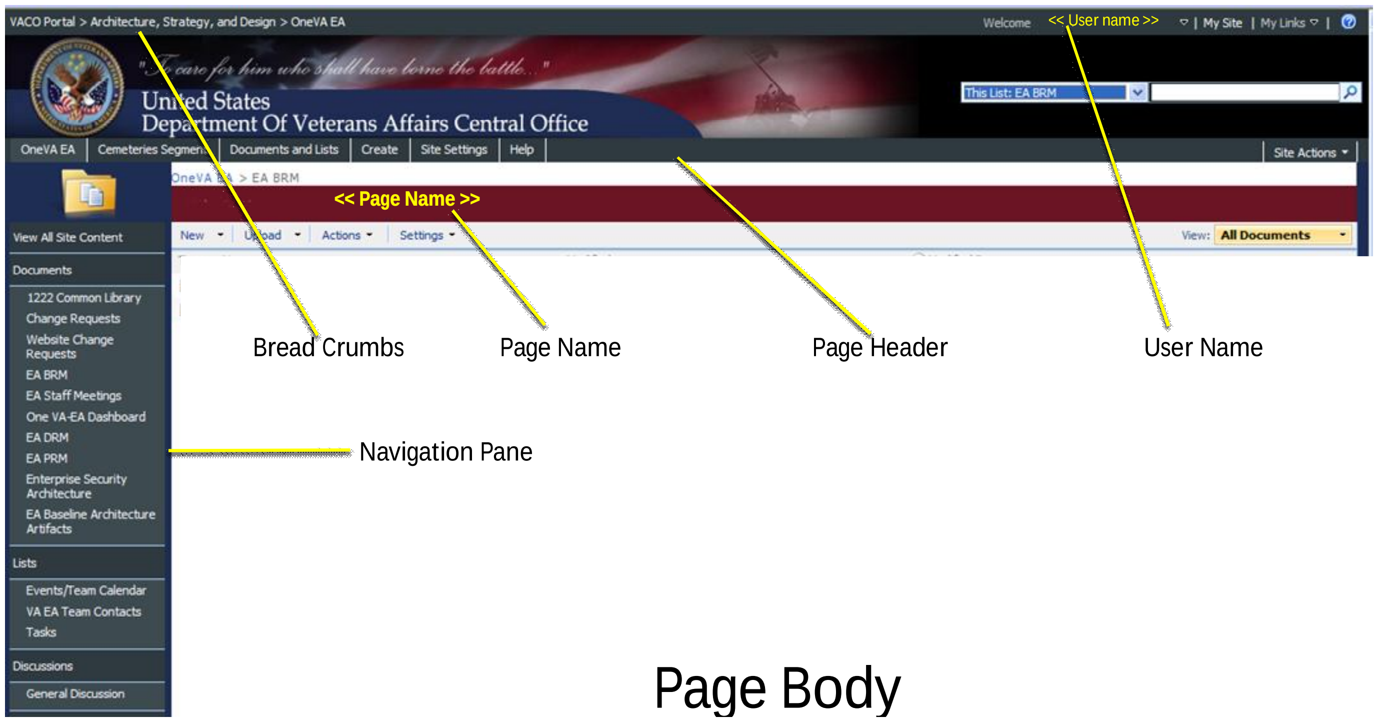

VACO Portal > Architecture, Strategy, and Design > OneVA EA
Welcome << User name >> | My Site | My Links |
"To care for him who shall have borne the battle..."
United States Department Of Veterans Affairs Central Office
This List: EA BRM
OneVA EA | Cemeteries Segment | Documents and Lists | Create | Site Settings | Help
Site Actions
OneVA EA > EA BRM
<< Page Name >>
New | Upload | Actions | Settings
View: All Documents
View All Site Content
Documents
1222 Common Library
Change Requests
Website Change Requests
EA BRM
EA Staff Meetings
One VA-EA Dashboard
EA DRM
EA PRM
Enterprise Security Architecture
EA Baseline Architecture Artifacts
Lists
Events/Team Calendar
VA EA Team Contacts
Tasks
Discussions
General Discussion
Bread Crumbs
Page Name
Page Header
User Name
Navigation Pane
Page Body

Figure 4 – Application Web Page Wire Frame Sample

2.3.3.2. **Colors**

The following principles affect the choice of colors for the web pages:

- **Accessibility -** avoid color combinations that would be indistinguishable for a user with color vision deficiencies.
- **Web safety -** avoid colors that would dither (i.e., an area of the color would be composed of pixels of different colors, which could reduce the visual quality of the graphic) when displayed on a Web browser.
- **Color** – use of color to convey information shall be accompanied by markup such as text weight, position, or shape. For example, the information that a piece of text is a link is conveyed by both blue text and underlines. Markup should be used to bring text formatting, padding, and block status, and sometimes color, such that it is used to define and structure content; thus bringing a section to the forefront.
- **UI** – No reference shall be made to colors in the UI. For example, if a Delete button were colored red, it would be inappropriate to provide users with instructions to click the red button; instead the instructions should refer to the Delete button.

Table 1 contains the color palette that can be used in a manner that complies with accessibility requirements and should form the basis for the look and feel design for each platform or product. Further colors may be documented in the individual style guide for each platform and product.

The background color will be white. RGB = (255, 255, 255); Hex = #FFFFFF.

The non-link text color will be black, except for situations that require other color as specified in these standards. RGB = (0, 0, 0); Hex = #000000.

Table 1 - VA Color Palette

	Hue	Sat	Lum	R	G	B	Hex
VA Blue	149	114	54	30	45	84	042D54
VA Maroon	233	164	60	107	20	36	041424
VA Medium Green	132	34	67	61	75	81	014B51
VA Dark Green	132	37	51	46	58	63	153A3F
VA Light Grey	160	0	228	242	242	242	F2F2F2
VA Medium Blue	142	240	201	173	209	255	ADD1FF

2.3.3.3. **Page Resolution**

Recommended target screen resolution for which applications should be designed is 1280 x 1024 for internal apps and 1024 x 768 for external applications. In order to provide maximum use of the screen area, web pages should be usable at higher resolutions (e.g., 1280x1024, etc.). It is recommended that pages be designed in such a way that they will “expand” to fit the user’s display resolutions.

2.3.3.4. **Navigation**

Navigation should be from top left to bottom right within a group box or dialog box. A rigid flow should not be enforced by enabling and disabling controls. This reduces the flexibility with which users can complete a task.

The navigation strategy should be designed so that users can navigate easily through the screens or pages, and so that navigation is separate from presentation and UI processing. Navigation links and controls should be displayed in a consistent way throughout your application to reduce user confusion and hide application complexity.

Design guidelines for page navigation include:

- Well-known design patterns are to be used to decouple the user interface from the navigation logic where this logic is complex
- Design tool-bars and menus to help users find functionality provided by the UI.
- Consider using wizards to implement navigation between forms in a predictable way.
- Determine how to preserve navigation state if the application must be preserved this state between sessions.
- Ensure that main navigation links are clearly and consistently labeled
- Ensure that navigation is easy to use for target audience
- If images, Flash, or DHTML is the main navigation, clear text links are in the footer section of the page (accessibility)
- Ensure navigation is structured in an unordered list (accessibility)
- Ensure navigation aids, such as site map, skip navigation link, or breadcrumbs are used (accessibility)
- Ensure that all navigation hyperlinks "work" — i.e., are not broken

2.4. Design Patterns

2.4.1. User Interface Elements

Table 2 - UI Component design guidelines

Design Rule Number	Design Rule Description
1	All new applications will comply with the Electronic and Information Technology Accessibility (EITA) Standards.
2	Display the results of actions immediately whenever and wherever possible.
3	Tool tips will be used whenever and wherever possible.
4	Tool tips will be used on all required fields.
5	Tool tips will not exceed two sentences.
6	The status bar will not display user-critical information.
7	If used, a Frequently Asked Questions (FAQ) section will be accessed from the application's Help menu.
8	When displaying results of a query, navigational controls will be provided. At a minimum, these controls will enable users to move to the previous and next pages.
9	Navigate from top left to bottom right within a group box or dialog box. Do not enforce a rigid flow by enabling and disabling controls. This reduces the flexibility with which users can complete a task.
10	When present, navigation menu bars and menu commands must appear in the same sequence across applications. Standardize the placement of organization-specific menus or menu items when they are used in multiple applications targeted to the same user group(s).
11	The navigation menu bar will be located at the top of a Web page in the lower right corner of the title block.
12	If users have to scroll (vertically) more than the equivalent of ten text lines to see all page content, navigation capability will be located at both top and bottom of the page.
13	If used, a contents menu (left-side list of navigational links) will be used only on the home page of an application, not on internal pages.
14	There will be no replication between the contents menu and navigation menu bar (drop-down menu bar) if they reside on the same page.
15	Menus will drop down when users click on the menu name in the Navigation Menu Bar.
16	Menus will be displayed with a different background color from the Web page background to distinguish from background text.

Design Rule Number	Design Rule Description
17	A Web-based navigation menu bar will include, at a minimum, the following items: Home—the Home menu option will return users to the applications' home page from anywhere within the application. Help—the Help menu option will include, at a minimum, the following items: – Help—Help at the application level. – About—the About menu option will be used to provide the following information about the application. ▪ Point of contact for that application (business owner) ▪ Help Desk phone number ▪ Application information ▪ Version information System Menu—the System menu will include a logoff option.
18	A menu option will fit on one line (i.e., not wrap to two or more lines). Therefore, the drop-down menu will be as wide as the longest menu command.
19	All input data must be validated when and where appropriate.
20	For errors flagged during validation, the fields in error will remain unchanged.
21	The home page will contain the following: • Title block • Application-specific instructions or items of note
22	The target screen resolution for which applications will be designed will be 800x600 pixels minimum size. They should never employ horizontal scrolling, vertical scrolling should be limited to a maximum of three pages deep. **Recommendation:** Applications should be usable at resolutions higher than the target resolution (e.g., 1024x768 pixels, 1280x1024 pixels, etc.). The page design should be liquid and resize to fit whatever size the browser window (within reason) that the user has available.
23	The default cursor, when not hovering over or clicking on/off a link, will be the standard arrow.
24	When "hovering" over an item that is a hot link, the cursor will be the "hand."
25	When the user has initiated a process, the "hour glass" cursor will be displayed for the duration of the process's execution.
26	Animation will not be used within the applications.
27	The title block is required and will contain the following: • Application name/title • VA logo and application logo • Date and time • Navigation • Menu bar
28	All application logos must be approved by the application owner and DCIO for systems development.

Design Rule Number	Design Rule Description
29	**Gateway page:** The official VA logo will be located in the upper left corner as a static graphic (no hyperlinks or animation). **Menu page or start page:** Each application logo will be used as a hyperlink or button image to go to the home page for that application. **Application home page:** The application logo will be located in the upper left corner as a static graphic (no hyperlinks or animation).
30	Logos will not behave in any way other than as static images.
31	A red asterisk will appear at the end of the field label after the colon for all required fields.
32	An explanation of the purpose of the red asterisk will appear on the start page of an application, as well as in the online help and the user references.
33	If a field requires one specific format, the required format will be clearly indicated to the users. When a field may have multiple formats, a help or tool tip will be used, depending on size.
34	Do not knowingly build non-functioning elements into the interface. Do not display messages (for example, “Under Construction,” “Future Use,” or “Not available”) as placeholders for non-functioning elements.
35	Cancel operation will stop running that operation in the background and return control of the application to the user. Recommendation: There should be a Cancel button to enable users to cancel (stop) long-running operations.
36	When used, the Cancel button will be located in proximity to the action to be cancelled.
37	The background color will be white. RGB = (255, 255, 255); Hex = #FFFFFF. The non-link text color will be black, except for situations that require other color as specified in these standards. RGB = (0, 0, 0); Hex = #000000.
38	All field labels will be presented in bold typeface and followed by a colon.
39	Text hyperlinks will be blue (RGB = 0, 0, 255; Hex = #0000FF) and underlined Recommendation: When most of the text on a page is hyperlinked, omit the underlines.
40	Data entry boxes will have the following attributes: • White background • Black border • Black text on entry
41	A data entry field will not exceed its required length, e.g., if the field can be ten characters in length, the input box must accept up to ten characters.
42	A progress indicator should be displayed for actions that could potentially take a long time to run (e.g., more than 5-10 seconds).
43	All standard pushbuttons will be labeled and behave according to standards listed in Table 2

Design Rule Number	Design Rule Description
44	Cancel/close buttons will be the rightmost element. Recommendation: If used, pushbuttons that are applicable to the entire page should be located on the bottom right of a Web page. Recommendation: If used, pushbuttons that are applicable to a group of page elements should be located in close proximity to those elements.

2.4.2. Standard Form Elements

2.4.2.1. Date Picker

Purpose:

The user finds or submits information based on a date or date range.

Description:

User can enter date in the text field or clicking on calendar icon. A calendar icon should be placed next to the field used for inputting date. The data format should be MM/DD/CCYY.

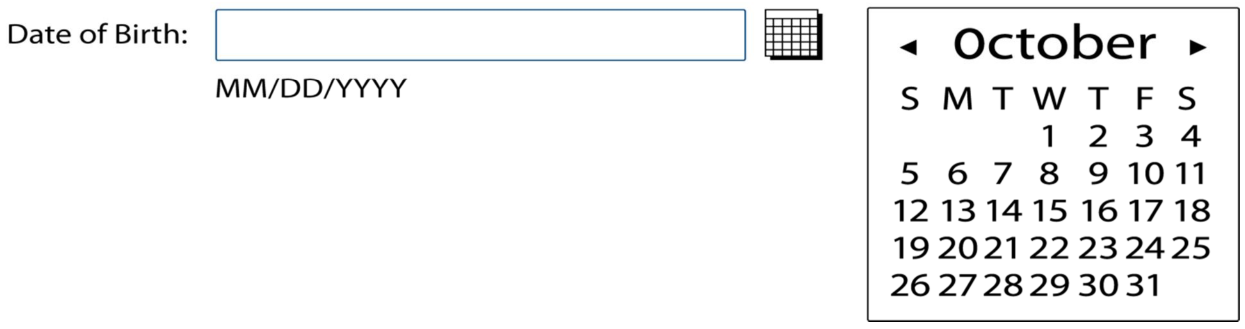


Figure 5 - Date Picker

The best practices for using Date Picker component are;

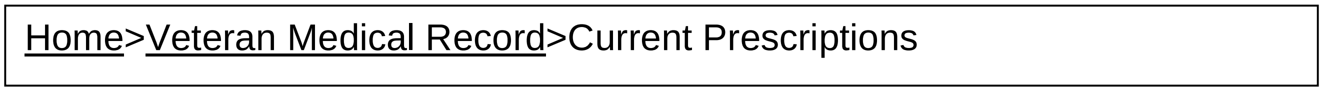
- A date field should be a single text field, and not separate fields for months, days, and years. This allows for internationalization and translation into multiple date formats.
- Inline Hint text should appear on the first date field of a page, indicating the format required to correctly populate the field.

- If the user enters incorrectly formatted data in the date field, field format validation should be handled through a simple JavaScript message window. Both the secondary window and inline date picker do not conflict with Accessibility guidelines, since the Date Picker is just an alternative method for entering a date.
- The date field is always present and allows direct manual input, so the user does not have to access the date picker.
- In Accessibility Mode, the inline date picker is not available, since a calendar view is not necessary for non-sighted users.
- Date pickers per field are still available. Each item in both the inline and secondary window date picker is keyboard traversable.
- The user can use the Tab key to move through the date picker, right to left and top to bottom, and Shift-Tab to move backwards.
- Selection is made with the Enter key.

2.4.2.2. **Breadcrumbs**

Purpose:

The user needs to know his location in the website's hierarchical structure in order to possibly browse back to a higher level in the hierarchy.



Home>Veteran Medical Record>Current Prescriptions

Figure 6 - Breadcrumbs

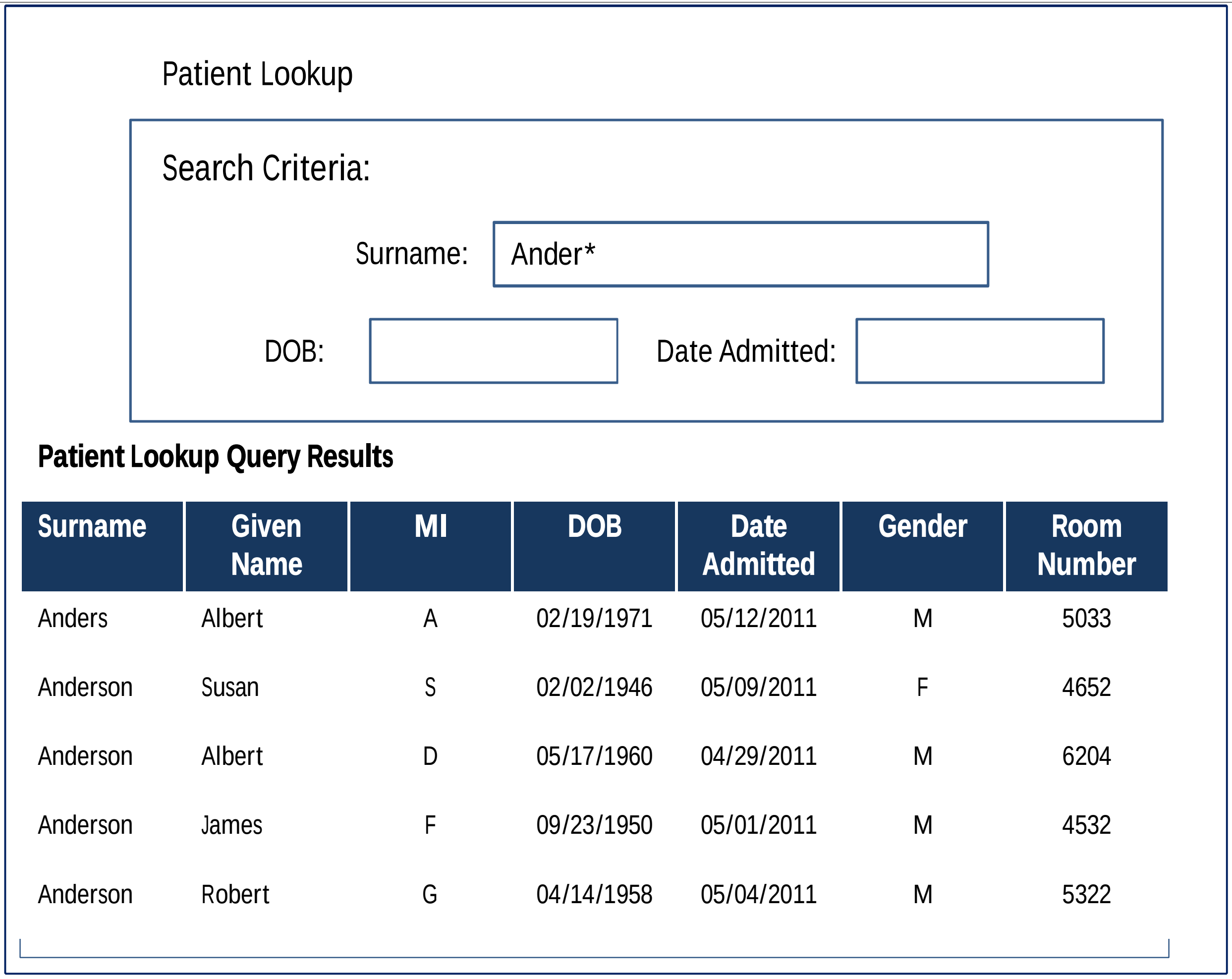
Description:

This pattern is used when the structure of the website follows a strict hierarchical structure of similar formatted content. The labels of the sections in the hierarchical path are shown to lead to the viewed page. Each label of the higher level subsections have links that lead to the respective section of the site. The label of the current page is at the end of the breadcrumb and is not linked. Each label is parted with a separating character. Popular characters are “»” or “>”. The separating characters and the spaces between the links and the labels are not linked. The structure of the website is more easily understood when it is laid out in a breadcrumb than if it is put into a menu.

2.4.2.3. **Search Box**

Purpose:

Search box is used when the users need to find an item or specific information.



Patient Lookup

Search Criteria:

Surname: Ander*

DOB:

Date Admitted:

Patient Lookup Query Results

Surname	Given Name	MI	DOB	Date Admitted	Gender	Room Number
Anders	Albert	A	02/19/1971	05/12/2011	M	5033
Anderson	Susan	S	02/02/1946	05/09/2011	F	4652
Anderson	Albert	D	05/17/1960	04/29/2011	M	6204
Anderson	James	F	09/23/1950	05/01/2011	M	4532
Anderson	Robert	G	04/14/1958	05/04/2011	M	5322

Figure 7 - Search Box

Description:

The search results are presented on a new page with a clear label containing at least "Search Results" or similar. A keyword search uses one or more complete words that are contained anywhere in the item's record, including: titles, notes, abstracts, summaries, descriptions and subjects.

The best practices for using a Search box component are;

- Provide only one type of quick search in an application.
- Make the Search button the right-most item among the quick search elements
- If a user clicks the Search button with the default value "- Search -" in the drop-down list box or with the text field empty, show one of the following error alerts, in a JavaScript pop-up window, depending on what was not specified:

  a) "Enter your search query in the text field"

  b) "Select an item in the drop-down list box" or

  c) "Enter your search query in the text field and select an item in the drop-down list box"

2.4.2.4. **Progress Bar**

Purpose:

A Progress Bar is used during long-running operations, to give a visual feedback to the user about the operation status.

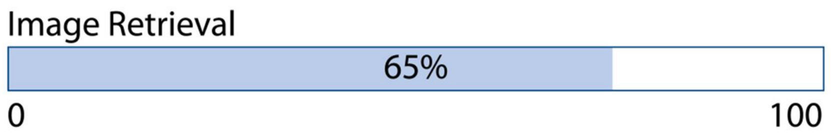


Figure 8 - Progress Bar

Description:

A progress bar keeps users informed as to how far they have progressed in the workflow. A progress bar (or progress meter) is a persistent navigation bar displaying a sequence of steps and highlighting the current step and optionally the degree or percentage of completion so far.

The progress bar will begin as soon as the user decides to start the process.

The final step in the progress bar should reflect the last screen where action is required (e.g., Complete Registration, Submit Order).

2.4.2.5. **Validation Messages**

Purpose:

A validation message is displayed, if the user entered data is not accurate.

Description:

User-Entered data requires validation and "sanity checking" before being submitted to the system. Users may not always enter required information in an expected format (e.g., a phone number containing non-numeric characters, spaces). When data is not checked on entry, validation errors may cascade into further corruption and it takes longer to detect and correct once entered. Further, if an invalid field eventually requires an entire page of information to be re-entered, the user will be required to perform much more work.

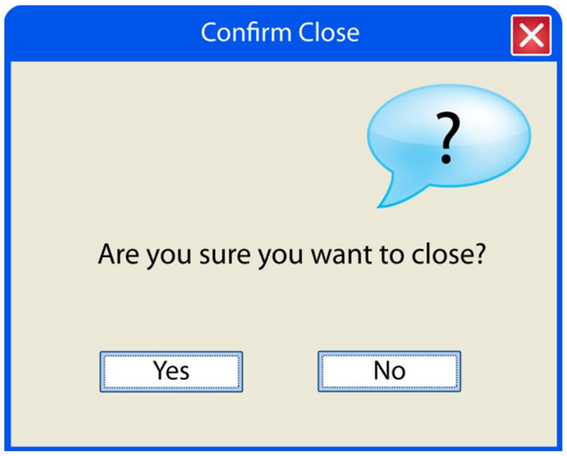


Figure 9 - Modal Dialog Box

2.4.2.6. **Modal Dialog Box**

Purpose:

A modal dialog box is used when the user is required to supply information before allowing the application to continue.

Description:

Applications use modal dialog boxes in conjunction with commands that require additional information before they can proceed. Modal dialog boxes are used when interaction with the application cannot proceed while the dialog box is displayed. For example, a progress dialog box that appears while the application is loading its data should be a modal dialog box. Modal dialog windows are used to collect information needed to complete a previously initiated command.

- Modal windows should not be sizable and therefore have a thin frame surrounding the entire window. Inside this frame, thick border should be drawn in the same color as the window caption. This border is a visual cue to the user that the application is in a modal state.
- Modal windows should not have a minimize button or a maximize button.
- A modal dialog should never contain a menu bar and its caption should be identical to the menu item that brings it up.
- All modal dialogs should contain an "OK" command button and a "Cancel" command button. The "OK" button should complete the action. The "Cancel" command should exit the dialog without completing the action.

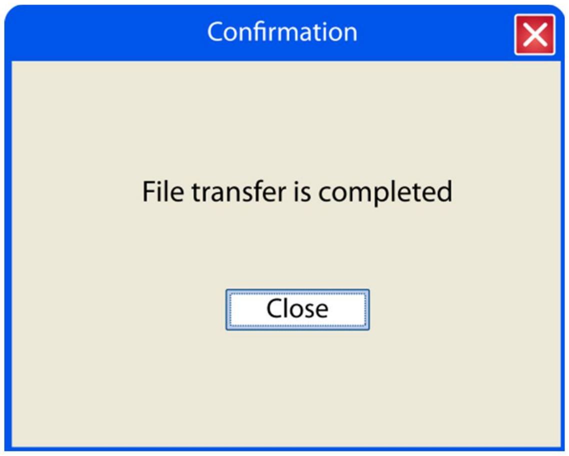


Figure 10 - Modeless Dialog Box

2.4.2.7. **Modeless Dialog Box**

Purpose:

Non-modal or modeless dialog boxes are used when the requested information is not essential for the user to continue. The window can be left open while the user continues to work on other functionality.

Description:

A dialog box allows the user to switch to another window while the dialog box is still on the screen. Use modeless dialog boxes whenever possible. The order in which users perform tasks might vary, or users might want to check information in other windows before dismissing the dialog box. Users might also want to go back and forth between the dialog box and the primary window.

- Modeless dialog windows should avoid using buttons labeled "OK" or "Cancel".
- Modeless dialog windows should include an action button labeled with a verb reflecting what the dialog does and a "Close" button.

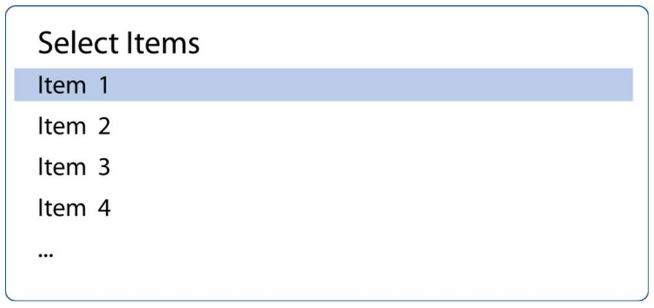


Figure 11 - List Box

2.4.2.8. **List Box**

Purpose:

The list box allows the user to select one or more items from a list contained within a static, multiple line test box.

Description:

In list box, the selections are always displayed. A list box can be set for multi-select, to allow users to choose several options at once. Since list boxes take up more space than combo boxes, they are most suitable when there is a limited number of choices and there is sufficient space on the form, or the multi-select option is needed. User can select only selections displayed in the list box and user cannot enter new data.

2.4.2.9. **Checkboxes & Radio buttons**

Purpose:

Check boxes are used to provide the selection of options that can be combined. Radio buttons are used when there is a list of two or more options that are mutually exclusive and the user must select exactly one choice. Clicking a non-selected radio button will deselect whatever other button was previously selected in the list.

Description:

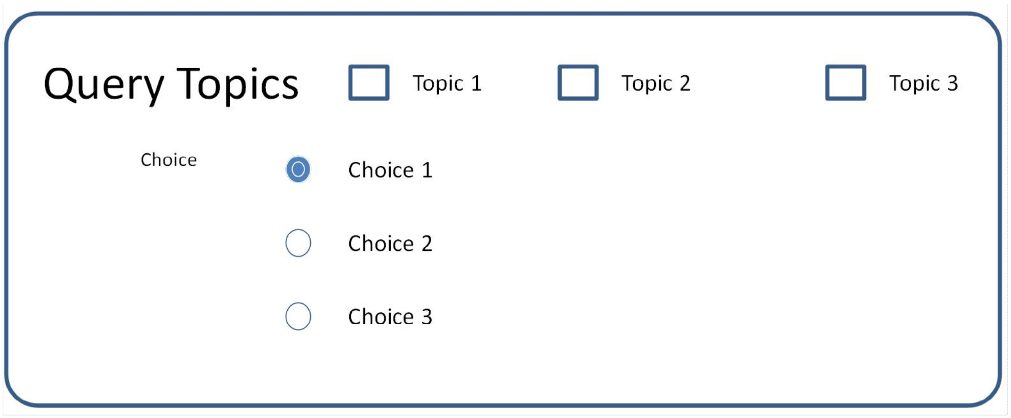


Figure 12 - Checkboxes & Radio buttons

A checkbox should be a small square that has a checkmark or an X when selected. Limit the number of check boxes in the user interface. A radio button should be a small circle that has a solid circle inside it when selected.

- Always use the affirmative choice for the check box label and keep the label brief to conserve space.
- Use positive and active wording for checkbox labels, so that it's clear what will happen if the user turns on the checkbox.
- Place a label to the right of the checkbox.
- Allow users to select a checkbox by clicking on either the box itself or its label.
- Use a checkbox only if ALL of the following are true:
- Users can choose between True and False, or between one value and no value.
- *Checked* means definitely True.
- *Unchecked* means definitely False, and never means *null*, *undefined*, *varies*, etc.
- The variable being represented is a Boolean, with only True or False values possible, never with a third, *null*, or *undefined*, or *varies*, etc., possible value.
- A user clicking the checkbox twice will return the UI to its previous state, as if the user did nothing.
- Do not use Checkboxes to trigger immediate actions (i.e. submitting the page, hyperlinking, or opening a window) unless the action is solely for purposes of redrawing the same page to facilitate

interactivity (i.e. showing additional information) on the same apparent page whose edits remain uncommitted and in-progress (from the user's perspective).

- Place each radio button label to the right of the radio button
- Allow users to select a radio button by clicking on either the button itself or its label
- A horizontal layout must be used with multiple options per line and make sure to space the buttons and labels so that it's abundantly clear which choice goes with which label.
- A default selection for radio buttons should be offered always**.** By definition, radio buttons always have exactly one option selected, and therefore shouldn't display them without a default selection.
- If users might need to refrain from making a selection, a radio button should be provided for this choice, such as one labeled "None."
- Access keys should be used for frequently used checkboxes and radio buttons , so users can quickly select their preferred options from the keyboard. This enhances accessibility for users with disabilities.
- Checkboxes and radio buttons should be used only to change settings, not as action buttons that make something happen.
- If the user clicks the Back button, any changes made to checkboxes or radio buttons on the page should be discarded and the original settings reinstated.

2.4.3. **Rich Client Components**

2.4.3.1. **Tab View**

Purpose:

The tabs are used to separate categorized information.

Description:

The tabs are shown high up the page and the area underneath it is connected to it visually. The information place in the tab pane belongs to the selected tab and can have its own sub-navigation. The currently selected category is highlighted by using contrasting color, shape, size, or typeface.

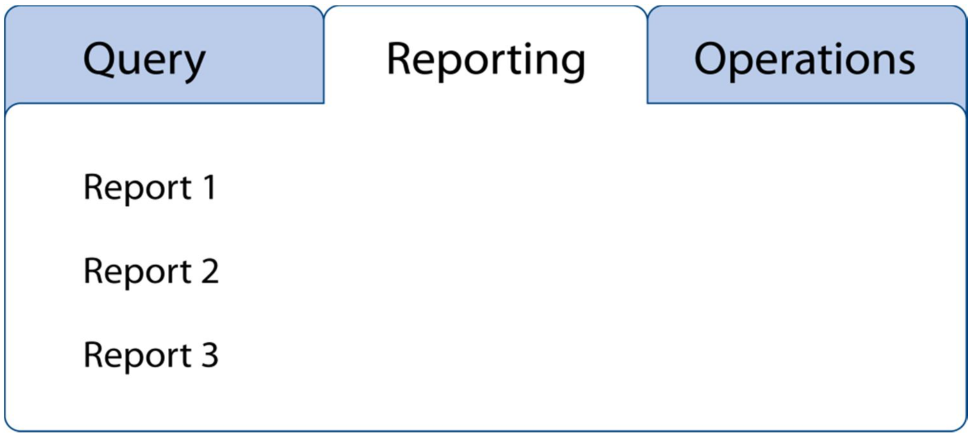


Figure 13 - Tab View

The best practices for using a Tab View component include:

- Tabs are used to alternate between views within the same context and the currently selected tab is highlighted. In addition to highlighting, the current tab can be marked by size, a boldfaced label, an icon, or by making it appear to be in front of the other tabs.
- The unselected tabs should be clearly visible and readable, reminding the user of the additional options. If the non-highlighted tabs are faded too much into the background, there's a risk that users will never click them and never discover the many hidden features.
- The labels should be short and concise.
- When the user clicks on a new tab the corresponding panel should be brought to the front immediately.

2.4.3.2. **Sliders**

Purpose:

A slider is a control that is used to select a value from a continuous or discontinuous range.

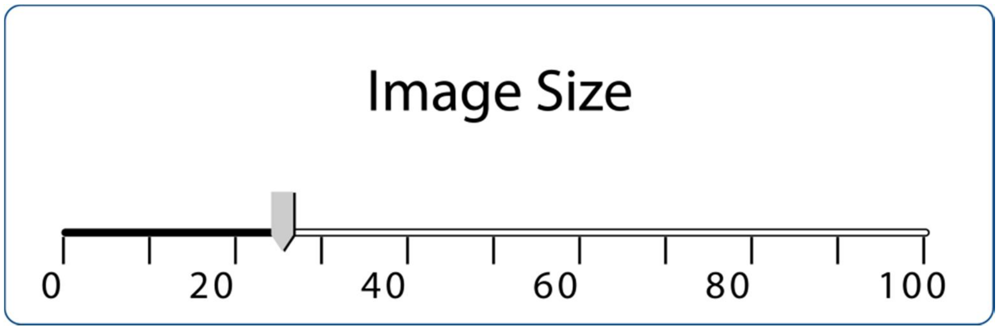


Figure 14 - Sliders

Description:

The position of the indicator reflects the current value. Major tick marks indicate large divisions along the range of values (for instance, every ten units); minor tick marks indicate smaller divisions (for instance, every five units).

- Make sure these range identifiers are descriptive and parallel. For example, use Low/High, Soft/Loud, and so on.
- Position a slider label either to the left of the slider or above and aligned with the left edge of the slider (or its range identifier, if present).
- When the slider represents actual values (such as screen resolution)—as opposed to relative values (such as lower or higher volume)—display the value of current selection underneath the slider.
- Center the text relative to the control, and include the units

2.4.4. **Functional Patterns**

2.4.4.1. **Intercepting Filter**

The Presentation Sub-Layer receives many different types of requests, which require varied types of processing. Both preprocessing and post-processing of a client Web request and response are required.

When a request enters a Web application, it often must pass several entrance tests prior to the main processing stage. The solution is to create pluggable filters to process common services in a standard manner without requiring changes to core request processing code. The filters intercept incoming requests and outgoing responses, allowing preprocessing and post-processing. We are able to add and remove these filters unobtrusively, without requiring changes to our existing code. Common types of filters are:

- **Base Filter –** serves as a common superclass for all filters. Common features can be encapsulated in the base filter and shared among all filters.
- **Standard Filter –** is controlled declaratively using a deployment descriptor, as described in the servlet specification version 2.3. The servlet 2.3 specification includes a standard mechanism for building filter chains and unobtrusively adding and removing filters from those chains. Filters are built around interfaces, and added or removed in a declarative manner by modifying the deployment descriptor for a Web application.
- **Custom Filter –** is implemented via a custom strategy defined by the developer. This is less flexible and less powerful than the preferred Standard Filter Strategy. The Custom Filter Strategy is less powerful because it cannot provide for the wrapping of request and response objects in a standard and portable way. Additionally, the request object cannot be modified, and some sort of buffering mechanism must be introduced if filters are to control the output stream.
- **Template Filter –** uses a base filter from which all others inherit and allows the base class to provide template method [GoF] functionality. In this case, the base filter is used to dictate the general steps that every filter must complete, while leaving the specifics of how to complete that step to each filter subclass. Typically, these would be coarsely defined, basic methods that simply impose a limited structure on each template. This strategy can be combined with any other filter strategy, as well.

Advantages:

- **Centralizes control with loosely coupled handlers** - Filters provide a central place for handling processing across multiple requests, as does a controller. Filters are better suited to massaging requests and responses for ultimate handling by a target resource, such as a controller. Filters allow for loosely coupled handlers, which can be combined in various combinations.
- **Improves reusability** - Filters promote cleaner application partitioning and encourages reuse. These pluggable interceptors are transparently added or removed from existing code, and due to their standard interface, they work in any combination and are reusable for varying presentations.
- **Improved consistency –** Reuse of common filters rather than writing custom editing routines for each input screen ensures that the same edits are used for each screen
- **Declarative and flexible configuration** - Numerous services are combined in varying permutations without recompiling of the core code base.

Disadvantage:

- **Information sharing is inefficient** - Sharing information between filters can be inefficient, since by definition each filter is loosely coupled. If large amounts of information must be shared between filters, then this approach may prove to be costly.

2.4.4.2. **Front Controller**

The Presentation Sub-Layer must control and coordinate processing of each user across multiple requests. The system requires a centralized access point for Presentation Sub-Layer request handling to support the integration of system services, content retrieval, view management, and navigation. When the user accesses the view directly without going through a centralized mechanism, problems may occur:

- Each view is required to provide its own system services, often resulting in duplicate code.
- View navigation is left to the views. This may result in commingled view content and view navigation.
- Distributed control is more difficult to maintain, since changes will often need to be made in many places.

The solution is to use a controller as the initial point of contact for handling a request. The controller manages the handling of the request, including invoking security services such as authentication and authorization, delegating business processing, managing the choice of an appropriate view, handling errors, and managing the selection of content creation strategies. Typically, a controller coordinates with a dispatcher component. Dispatchers are responsible for view management and navigation. Thus, a dispatcher chooses the next view for the user and vectors control to the resource. Dispatchers may be encapsulated within the controller directly or can be extracted into a separate component.

There are several strategies for implementing a controller.

- **Servlet Front Strategy** - This strategy is preferred to the JSP Front Strategy. The controller manages the aspects of request handling that are related to business processing and control flow. These responsibilities are more appropriately encapsulated in a servlet rather than in a JSP page.
- **JSP Front Strategy -** This strategy is more cumbersome when the complete cycle of coding, compilation, testing, and debugging is considered.
- **Command and Controller Strategy** - Based on the Command pattern, this strategy provides a generic interface to the helper components to which the controller may delegate responsibility, minimizing the coupling among these components. This strategy also facilitates the creation of composite commands.
- **Base Front Strategy** - Used in combination with the Servlet Front Strategy, this strategy implements a controller base class, whose implementation other controllers may extend.
- **Filter Controller Strategy -** Filters provide similar support for centralizing request processing control.

Advantages:

- **Centralizes Control** - A controller provides a central place to handle system services and business logic across multiple requests. A controller manages business logic processing and request handling. Centralized access to an application means that requests are easily tracked and logged.
- **Improves Manageability of Security** - A controller centralizes control, providing a choke point for illicit access attempts into the Web application. Auditing a single entrance into the application requires fewer resources than distributing security checks across all pages.
- **Improves Reusability** - A controller promotes cleaner application partitioning and encourages reuse, as code that is common among components moves into a controller or is managed by a controller.

2.4.4.3. **Application Controller**

The Application Controller pattern manages multiple user interactions by providing a centralized point for handling screen navigation and the flow of an application. The Application controller enforces the flow of control between different views and gain state management across these views.

Some of the common strategies to implement the Application Controller pattern are:

- **Command Handler Strategy**

An Application Controller that obtains and invokes Command objects is also referred to as a Command Handler. A Command Handler manages the life cycle of these command objects, which support an application's use cases and functions within the scope of a single request rather than across multiple requests. Commands encapsulate use case-specific processing logic and are only loosely coupled with the object that invokes them.

- **View Handler Strategy**

An Application Controller that obtains and invokes view-related objects is referred to as a View Handler. The Application Controller resolves and dispatches to the appropriate view, an activity referred to more generally as view management. The value of centralized view management is improved modularity, extensibility, maintainability, and reusability.

- **Transform Handler Strategy**

An Application Controller that obtains and invokes view-related objects for use with a transformation engine is referred to as a Transform Handler.

- **Navigation and Flow Control Strategy**

Users typically need to move through application screens in a particular order. This flow through the application can be controlled in several ways. The most basic way is simply to check whether a particular precondition has been satisfied before allowing access to a certain page. If the flow order of the screens depends on the current state of parts of the application then the flow control can leverage a simple state-machine. These rules can then be made declarative so that this strategy can also help reduce the coupling between the code and the navigation/flow control constraints. Finally it can also be used for limiting duplicate requests.

- **Message Handling Strategies**

An Application Controller is commonly used in the Presentation Sub-Layer to perform action and view management along with a Front Controller. However, an Application Controller can also be used in other Sub-Layers and in other contexts. For example, an Application Controller might also support Web service requests that require routing and action management.

- **Custom SOAP Message Handling Strategy**

A Front Controller typically acts as the central access point for SOAP requests. In this strategy, the controller delegates to an Application Controller, which uses a Custom SOAP Filter to perform the pre and post processing of the message. The Application Controller then performs action and view management, including the appropriate validation and error handling. The "view" is, in effect, simply the response that is prepared for return to the client.

- **JAX RPC Message Handling Strategy**

A Front Controller typically acts as the central access point for SOAP requests. In this strategy, the controller delegates to an Application Controller, which uses a JAX-RPC Filter strategy to perform the pre and post processing of the message. The Application Controller then performs action and view management, including the appropriate validation and error handling. The "view" is, in effect, simply the response that is prepared for return to the client.

The Application Controller can be used in applications with deterministic flow between views. It provides following benefits:

Advantages:

- **Improves code modularity and maintainability –** Easy to extend the application and easy to test discrete parts of the request-handling code independent of a web container.
- **Improves code reusability –** Reuse action and view-management code.
- **Improves request-handling extensibility –** Allows adding use case functionality to an application incrementally.

2.4.4.4. **Business Delegate**

Presentation Sub-Layer components can, in theory, interact directly with business services. This direct interaction exposes the underlying implementation details of the business service to the Presentation Sub-Layer. When the implementation of the business services change, the exposed implementation code in the Presentation Sub-Layer must change too. When Presentation Sub-Layer components use the underlying API directly, there may be a detrimental impact on network performance.

The solution is to use a Business Delegate to reduce coupling between Presentation Sub-Layer clients and business services. The Business Delegate hides the underlying implementation details of the business service. The Business Delegate acts as a client-side business abstraction. The delegate may cache results and references to remote business services which can significantly improve performance.

There are several strategies to implement a Business Delegate

- **Delegate Proxy Strategy** - The Business Delegate provides a proxy function to pass the client methods to the session it is encapsulating. The Business Delegate may additionally cache any necessary data, including the remote references to the session's home or remote objects to improve performance by reducing the number of lookups.
- **Delegate Adapter Strategy** - Disparate systems may use an XML as the integration language. Integrating one system to another typically requires an Adapter to meld the two disparate systems.

Pros:

- **Reduces Coupling, Improves Manageability** - The Business Delegate reduces coupling between the Presentation Sub-Layer and the Business Sub-Layer by hiding all Business Sub-Layer implementation details. It is easier to manage changes because they are centralized in one place, the Business Delegate.
- **Translates Business Service Exceptions** - The Business Delegate is responsible for translating any network or infrastructure-related exceptions into business exceptions, shielding clients from knowledge of the underlying implementation specifics.

- **Implements Failure Recovery and Thread Synchronization** - The Business Delegate on encountering a business service failure, may implement automatic recovery features without exposing the problem to the client.
- **Exposes Simpler, Uniform Interface to Business Sub-Layer** - The Business Delegate may provide a variant of the interface provided by the underlying enterprise beans.
- **Impacts Performance** - The Business Delegate may provide caching services (and better performance) to the Presentation Sub-Layer for common service requests.

Cons:

- **Introduces Additional Layer** - The Business Delegate may be seen as adding an unnecessary layer between the client and the service, thus introducing added complexity and decreasing flexibility.
- **Hides Remoteness** - Typically, a method invocations on the Business Delegate results in a remote method invocation under the wraps. Ignoring this, the developer may tend to make numerous method invocations to perform a single task, thus increasing the network traffic.

2.4.5. **Supported Protocols/Invocation Methods**

On the client layer, the user interface is rendered using HTML, with limited use of Asynchronous JavaScript and XML (AJAX). In most cases, AJAX can be used to improve usability of user interfaces by:

- Allowing dynamic rearrangement of the web content based on user input. For example, AJAX can display or hide certain form fields based on their relevance to the user work.
- Validation of user input within the browser.
- Providing assistance to the user through such techniques as context sensitive hints and automatic completion of text input.

Presentation Sub-Layer services can communicate with the ESB to access Business Sub-Layer services. Once the ESB is deployed, the Presentation Sub-Layer service will access services in the ESB. Until that time the Presentation Sub-Layer services can directly access in services in the Business Sub-Layer. For application specific services that are not to be shared beyond the applications, direct communication between the Presentation Sub-Layer services and Business Sub-Layer services.

The Strategy for communicating with the Business Sub-Layer should be decided based on the deployed model. The commonly used communication methods are;

- **Direct method calls:** If the Business Sub-Layer service and Presentation Sub-Layer service are part of the same application, direct method calls SHOULD be used, however message-based communication can be used. Within a single application, it is not necessary to use VA standard messages.
- **Web services:** If the Business Sub-Layer service is deployed by a separate application from the Presentation Sub-Layer service standards-based messages SHOULD be used be used for communication between the services.

2.5. Security

The Presentation Sub-Layer presents specific threats and risks to the security of the application and care must be taken to secure it against these threats. The application security must be approached from the earliest stages of the application design. This section describes how to assess and mitigate the security risks facing the application.

2.5.1. **Secure Communications**

Exchanging data over exposed networks, such as the Internet, introduces security risks. If the application processes either PII or PHI, the secrecy and integrity of the data as it travels between application components must be ensured. The approach to secure communications is by employing Transport Level Security (TLS) as per the NIST guidelines between the web browser on the end-user device and web server.

The Presentation Sub-Layer is a common place to implement cryptographic classes that support symmetric and asymmetric encryption, hashing, digital certificates, and key exchange. Communications between the browser and the web server should be performed using digitally signed messages. In addition, communications between the web server and the ESB will also use digitally signed messages. If the messages pass unchanged from the end-user device to the ESB they will be signed with the user's key. The Presentation Sub-Layer / Portal's key will be used to sign messages that are created or modified in the Presentation Sub-Layer.

2.5.2. **Authentication & Authorization**

The purpose of authentication is to securely establish the identity of a person who wants to use the application. Authorization and auditing require authentication. The most common authentication techniques such as Single Sign on(SSO), smart cards, biometrics, and digital certificates, are gaining in popularity as they become more accessible and easier to implement. This is especially true in applications that process sensitive PII or PHI. VA is in the process of selecting and implementing and enterprise SSO solution.

Authorization is the process of determining whether a user has permission to access a particular resource or piece of functionality. To perform authorization, an authentication mechanism must be placed to establish the identity of the user, and that mechanism must determine the identity of the user accurately and reliably.

Once the SSO is implemented users will be authenticated once when they start an online system regardless of the number of systems that are accessed. After being authenticated, the user will need to be authorized to access each application or resource. SSO improves efficiency by only requiring a single authorization regardless of the number of authorization decisions that may eventually need to be made.

Prior to the implementation of the SSO solution, each application will be responsible for the authentication and authorization of its own users. There are two ways to pass the results of the authentication decision to downstream applications. However, these applications will still need to make and log their own authorization decisions.

- **Trusted subsystem model**

The web application authenticates and authorizes the user at the first point of contact, and it creates a trusted identity to represent the user. Essentially the system that is the first point of the contact authenticates the user and then vouches for the user to all downstream applications. Further, the application that is the first point of contact may pass its own identity to the downstream applications and asks for work to be performed on its behalf rather than the user's behalf. This trusted identity is then used by the downstream applications or business services as the basis for their authorization decision. As attractive as this approach may seem, it has a major downside in that implementation of this approach is tantamount to the development of an SSO solution.

- **Impersonation/delegation model**

In the impersonation/delegation model, the Web application authenticates and authorizes the user at the first point of contact as before. However, these original credentials are flowed to downstream applications instead of passing the same trusted identity for all users. This enables downstream applications to perform their own authentication and authorization tests using the real security credentials of the original user.

2.5.3. **Applets / ActiveX controls**

As general guideline, usage of Applets/ActiveX controls is not recommended except in the cases where the local system resources needs to accessed (e.g., Device Integration) to meet the business requirements. Mobile code such as Applets/ ActiveX controls present security risks to both the web application and to the client system beyond those presented by simple web applications. Applets/ ActiveX controls present risks both to the user devices as well as opening vulnerabilities on the server. All Applets/ ActiveX controls will need to be digitally signed and checked for malicious code and known code vulnerabilities prior to being downloaded to the user device.

Approaches such as Code Access Security (CAS) allow code to be trusted to varying degrees, depending on factors such as where the code comes from and its strong assembly name. CAS enables to specify the operations the code can or cannot perform. CAS supports a permission support mechanism where code can explicitly request specific permissions and explicitly refuse others that it knows it never requires.

2.5.4. **Threat Prevention**

While exposing systems and applications through the web increases accessibility both internally and externally, it also brings with it the increased exposure and need for mechanisms and techniques for preventing threats to the systems and corresponding data. Preventing threats to web applications can be accomplished through various mechanisms depending on the specific type of threat. The first line of defense is the protections that are provided by the network engineering team which both provides protection and is transparent to the applications. Thus, these protections are normally in place with not effort on the part of the application designer or developer. Generally, network firewalls and intrusion detection systems are put in place at the network layer to prevent unauthorized access to resources within the organization.

Ensuring input validation as well as proper use and protection of headers, cookies, form fields, and URL parameters will also assist in the prevention of threats related to parameter manipulation. Input validation can and should be accomplished for known values, but the most issues arise from accepting unknown or dynamic values. Preventing parameter manipulation can be accomplished with various techniques including the use of encryption, session tokens, session variables, pattern matching, hashing and input validation. Input validation includes ensuring data input conforms to the following:

- Is strongly typed
- Is the correct syntax
- Is within length boundaries
- Is comprised of only legal characters
- Numbers are properly signed and within range boundaries
- Other defined validation policies (i.e. known malicious patterns, custom defined patterns)

While the Presentation Sub-Layer is an obvious location for data validation activities, it should not be the only layer in which data validation takes place. The Presentation Sub-Layer should be primarily concerned with input validation and not attempt to implement business rule validation as this would fall to the Business Process Sub-Layer.

2.5.5. **Validation**

The message validation logic enforces a well-defined policy that specifies which parts of a request message are required for the service to successfully process it. Any message that does not meet the criteria is rejected.

Problem:

- A Web service interacts with other applications over a network. Incoming data may be malformed and may have been transmitted for malicious purposes. There is also a risk of injection attacks, where data from incoming messages is tampered with to include additional syntax. How do you protect Web services from malformed or malicious content?
- **Solution:** Assume that all input data is malicious until proven otherwise, and use message validation to protect against input attacks, such as SQL injection, buffer overflows, and other types of attacks. The message validation logic validates the XML message payloads against an XML schema (XSD) to ensure that they are well-formed and consistent with what the Web service expects to process. The validation logic also measures the messages against certain criteria by examining the message size, the message content, and the character sets that are used.

Benefits:

- The Web service is protected from malformed and malicious content. This helps protect against injection attacks, even for Web services that do not implement access control.
- The Web service performs validation independently of the client. It does not accept messages simply because they have already been validated by the client.

Security Considerations:

- Message validation can help protect against denial of service attacks, but the message validation logic must be very efficient when it conducts its validation checks.
- Using a validating parser and verifying the input message against its XML Schema (XSD) results in a significant increase in CPU processing.
- Instead of building the message validation logic into the Web service itself, you can place it in an intermediary, such as a wire speed appliance. This allows several Web services to use the same intermediary, and it enables each Web service to dedicate its resources to processing legitimate messages. It also ensures that invalid messages never reach the Web service.
- XML message payloads that contain a CDATA field can be used to inject illegal characters that are ignored by the XML parser. If CDATA fields are necessary, they must be inspected for malicious content.
- The Web service may obtain data for response messages from external sources. There is no guarantee that external data sources properly validate data. Passing responses without message validation makes the Web service a potential "carrier" of malicious input from external data sources.

3. Business Sub-Layer

The Business Layer implements the core functionality of the system and encapsulates the relevant business logic. It manages business processing rules and logic and transforms inputs into outputs. It is typically composed of components and workflows which are exposed as services. The business services implement core business functionality that may include multiple tasks that must be performed in a specific sequence and which interact with each other to achieve the end result.

Communication between the services running in the Presentation Sub-Layer and the Business Sub-layer was discussed previously. The services in the Business Logic Sub-Layer also need to communicate with:

- **Legacy Systems through a Service Facade –** Business Sub-Layer services are able to access legacy systems functionality through service facades placed in front of those legacy systems. The service facade is built by the legacy application group and directly accesses the legacy application – usually providing access to a single legacy system transaction or a small group of related transactions. The service facade appears to the Business Sub-Layer services as an SOA service and to the legacy application like another legacy system or transaction user.
- **Data from Data Layer Services –** The Data Layer controls access to the data needed by the Business Sub-Layer Services. Data access services reside in the SOA Layer and communicate with the Data Layer services. The communication between the SOA Layer Data Access Services and the Data Layer services is discussed in Chapter

- Data Layer.

3.1. Architectural Considerations

One goal of designing a Business Sub-Layer service is to minimize complexity by separating the business logic – the processing of the business rules, from the control logic – the order in which the business rules are executed. Processing business rules is the task of the service while management of the workflow is business process orchestration. Following are architectural considerations for implementing VA Business Sub-Layer services.

- **Loose Coupling**
  - Must be loosely coupled both with presentation and data services as well as with other business services.
  - Must provide public interfaces that all service consumers can use to interact with Business Sub-Layer.
  - Must interface with Data Access Sub-Layer for processing the data logic and data manipulation.
- **High Cohesion**
  - Must contain only Business Sub-Layer functionality. Neither Presentation nor Data Access logic shall be mixed with business logic.
- **Security**
  - **Authentication –** Eventually authentication will be performed by the enterprise SSO capability. Until then individual applications and services will be responsible for performing ther own authentication. This is likely to include the authentication of the user in the Presentation Sub-Layer and the Presentation Sub-Layer requesting the Business Sub-Layer on it own on behalf of the ultimate end-user. This will result in a very small numbers of users of the Business Sub-Layer service.
  - **Authorization –** Authorization allows access to protected resources for authenticated users based on their identity, account groups, roles, or other contextual information.

Role-based authorization is the preferred approach. Services should avoid mixing authorization logic with business logic.

- **Transaction Management –** Orchestrations must perform transaction management. Transaction management must NOT be delegated to the Data Access Sub-Layer. The Business Sub-Layer shall control the transaction context i.e., where the logical unit of work begins and ends (when transactions are committed, rolled back, or compensating transactions initiated).
- **Reusability** – Must be designed for maximum code reusability. Each business function must be descriptive with clearly documented processing logic, inputs, outputs and exceptions; making it easier to be identified and used by an application or an external service. All business services must be entered into the appropriate service registries / repositories.
- **Scalability** – Should be designed for statelessness to facilitate scalability. Additional instances should be instantiated to scale out according to the increased load and leverage container clustering capabilities to handle failover.
- **Validation –** Must perform all input and method parameter validations within the Business Sub-Layer even if basic input validation has been performed in the Presentation Sub-Layer. Presentation Sub-Layer input validation will typically be limited to format (e.g., there are no non-numeric characters in the phone number) or range validations. Validation in the Business Sub-Layer will include much more detailed business rule validation e.g., does the Veteran's name and stated phone number agree with the values in VA records. Failure to validate data could leave the application vulnerable to data inconsistencies, business rule violations and security issues.
- **Testability** – All public methods of business functions must be testable. Unit test cases must be developed and provided for all public methods of the business Sub-Layer interfaces.
- **Exception handling**
  - Must provide an effective exception management strategy in the Business Sub-Layer. Raising and handling exception is an expensive operation therefore performance impacts must be considered. Must not catch exceptions that are generated elsewhere and which can be caught elsewhere.
  - Should not use exceptions to control program flow.
  - Should be logged prior to being passed to the users.
  - Exceptions must not reveal sensitive information.
- **Logging & Auditing**
  - Must use a centralized logging and auditing for all authentication and authorization decisions and each access to business functions and critical resources within the Business Sub-Layer.
  - Must not store business sensitive information in log files.
  - Must ensure that logging failure does not impact business process.

3.2. Business Layer Components

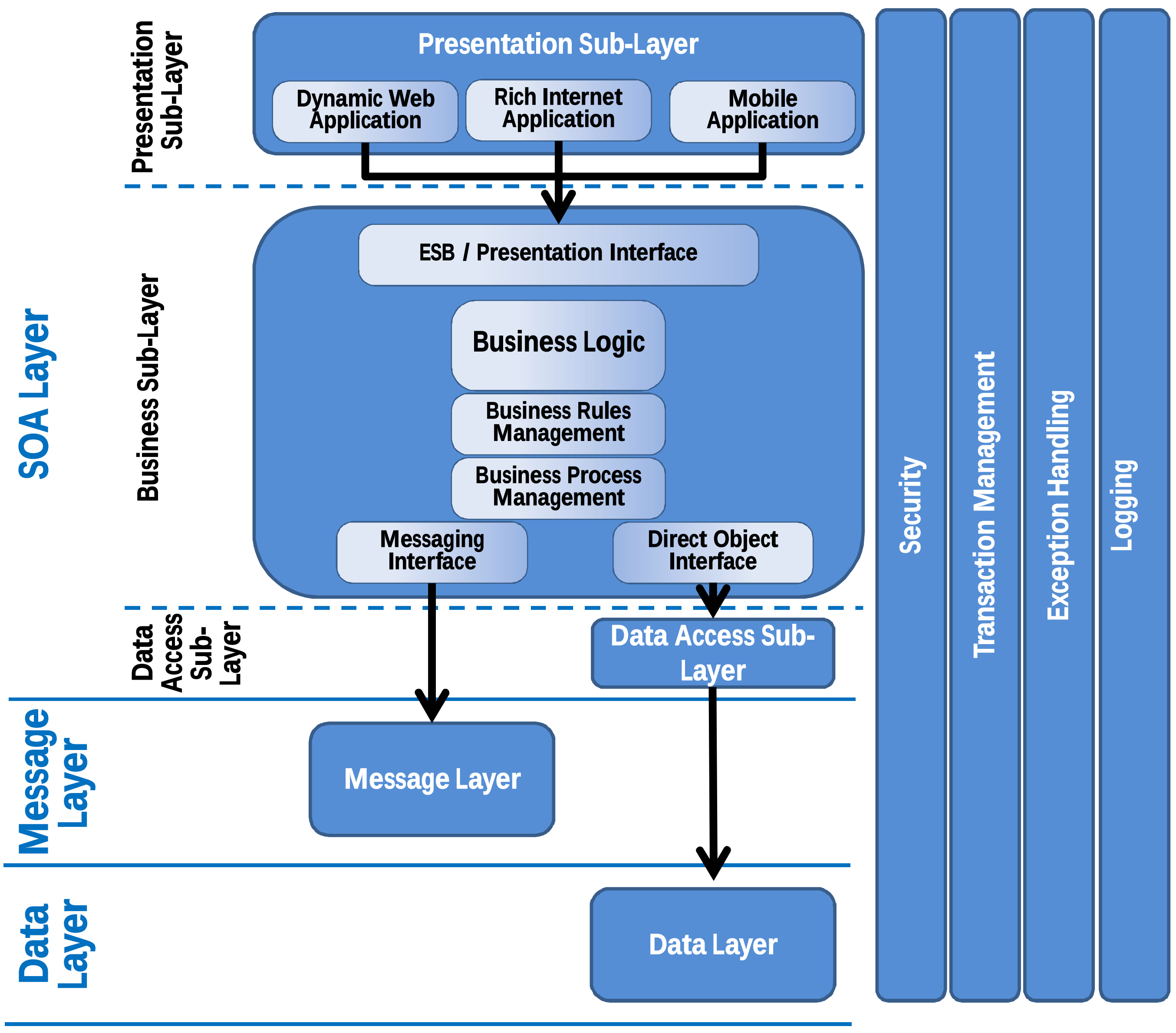


Figure 15 - Business Layer Components

3.2.1. **Business Sub-Layer Interfaces**

The Business Sub-Layer provides for the management and execution of business processes. The SOA services executing must interface with services running in the Presentation Sub-Layer and Data Layer. Eventually, the interface between the business services running in the SOA Business Sub-Layer and the presentation services running in the SOA Presentation Sub-Layer will be through the ESB, but in the interim, prior to the implementation of the ESB, there will be a direct interface between the business services and presentation services in their respective sub-layers. The interfaces between the business services running in the Business Sub-Layer of the SOA Layer will access the Data Layer data services through the Data Access Sub-Layer of the SOA Layer. A major advantage of separating presentation, business, and data services into different layers and sub-layers is that is assure loose coupling between the various types of services by isolating them to separate sub-layers of the architecture. The Business Sub-Layer also supports an interface to the Message Sub-Layer to allow Business Sub-Layer Services in one VA application to interface with business services in another VA application or with applications external to VA.

In addition to loose coupling between services running at different layers and sub-layers, there should be loose coupling between services running in a single sub-layer. Loose coupling between services can be maintained by using orchestration to control flow information between services rather than by imbedding control in the service. The more formal the communications between the service and the more control is externalized, the more loosely couple the services can be. Further, to allow reusability of services and for services to be used in as many orchestrations as possible, the services in the Business Sub-Layer should be a fine grained as possible.

3.2.1.1. **Business Sub-Layer – Presentation Sub-Layer Interface**

As noted above, eventually communications between the Presentation Sub-Layer and the Business Sub-Layer will be through the ESB because the business services will be inside the ESB security domain and with be required to go through the ESB security gateway. Until such time as the ESB security domain is established and the ESB is operational, communication between presentation and business services will be direct.

3.2.1.2. **Business Interfaces**

Business interface acts as a bridge between the Presentation Sub-Layer and Business Sub-Layer. Solutions developed using Plain Old Java Objects (POJOs) and Enterprise Java Beans (EJBs) are considered to be the best practice.

3.2.1.3. **Service Interfaces**

There will be more overhead when services communicate across layers and sub-layers of the architecture than when services communicate with in a sub-layer of the architecture. Once the ESB is in place there will be the need for messages from the Presentation Sub-Layer and ESB to be digitally signed and for the digital signature to be checked at the ESB. Because of the overhead communicating between sub-layers services that communicate across layers and sub-layers should be coarse grained so as to minimize the number times the boundary is crossed.

Communication between the Presentation and Business Sub-Layers should use common data formats for the interface schema that can be extended without affecting consumers of the service. Solutions designed based on SOA principles are the best fit for service interface. Typical solutions include SOAP or REST based Web Services.

3.2.2. **Business Sub-Layer Services**

3.2.2.1. **Business Logic**

Business Logic is the core of the application where the business capability is implemented. Any interaction with the Data Layer has to be through the Data Access Sub-Layer for enhanced security and enhanced separation between business logic and data logic. Business logic should be implemented as plain objects with coarse grained stateless functions, so that the business logic is easily exposable. It should perform transaction management to the extent any is needed beyond the capabilities provided by the orchestration tool that is being used and provide an effective exception management strategy.

3.2.2.2. **Business Process Management**

One of the key considerations for the Business Logic Sub-Layer is to “build for change”. Combining business rules management and business process management provides agility by allowing the decisions, and their underlying rules, to be changed independently from the processes and control logic, often in real-time by business managers.

Business Process Management’s (BPM’s) main goal is to facilitate Business Flow and Rules Change in an organization. Modeling of the business process can be achieved using standard notations based on Business Process Modeling Notation (BPMN[2]). The outcome of the process modeling phase is usually a pseudo-executable process flow that can be exported in an executable language e.g., Business Processing Execution language (BPEL) format supported by the BPM engine. Orchestration servers provide a runtime environment for executing BPEL code.

Since BPEL was designed specifically for the definition of business processes. It provides support for long running transactions, compensation, event management, correlation[3], etc. A BPM process can be invoked synchronously or asynchronously. BPM execution process can be extended to target standard Java Platform, Enterprise Edition (Java EE [J2EE]) artifacts such as EJB, JMS, RMI, JCA, JBI, and components developed using resources such as the Web Services Invocation Framework (WSIF).

BPM Best Practices:

- Should plan for business process maturity from definition of objectives and scope to initial deployment to sustaining the processes.
- Should define a methodology for analysing, designing, developing, and simulating processes or workflows.
- Should define and set minimum governance rules to ensure project consistency.
- Should prototype a process that is used broadly but has relatively low complexity.
- Should align with SOA objectives by creating a central repository to hold BPM assets/services.

3.2.2.3. **Business Rules Management**

A Business Rule Management System (BRMS) is a software system used to manage the business rules that includes policies, requirements, and conditional statements that are used to determine the actions that take place in the applications. Proper business rule and business process management require the management of both the logical specification of the rules and the processes as well as managing their physical instantiation.

Rules engines or inference engines can be used to execute business rules that have been externalized from application code. This externalization of business rules allows the business users to modify the rules frequently without the need of IT intervention. The system as a whole becomes more adaptable with business rules since the business rules can be changed dynamically, although QA and other testing would still be required.

[2] The recommended standard is BPMN 2.0

[3] When services and message processing is run asynchronously multiple service requests may generate messages requesting information from remote services or applications. When messages arrive with the results of the remote service processing it is not immediately clear to which service request they are a response. Correlation is the process of aligning service requests and service responses.

Use of the business rules and a business rules engine means that individual business rules can be changed with no impact to other existing business rules. While the new rules will need two go through quality assurance and testing there is no need to fully regression test the full application.

3.2.3. **Messaging Interfaces**

The Message Layer provides the use of VA Standard Messages for communicating between services in different applications or with services external to VA. Messages – although not necessarily VA Standard Messages, are used to ensure the loose coupling of services within an application. VA Standard Messages will be based on syntactically and semantically harmonized data. Because syntactic and semantic harmonization of all data will require some extended period of time, this will not be accomplished in the near term. Therefore, part of the development of messages to communicate between applications will require work to harmonize segments of VA data to allow the messages to be defined and used. Messages can either be synchronous of asynchronous, but asynchronous messages are preferred as synchronized messages required a synchronization mechanism that will need to be developed. The orchestration engine can be used to support message synchronization. Messages between services within an application should be Java-based.

3.3. Design Patterns

3.3.1. **Service Facade**

Problem:

- Until all VA systems are modernized and developed based on SOA services there will be some functionality and data that resides in the modernized systems and some in the legacy systems. It will also be the case that at times, parts of an application will be modernized and parts will be legacy and there will be a need for the capabilities and / or data to be used concurrently. There are two use cases 1) a modernized, service-based VA Business Application needs to access data in a legacy system data base or access system functionality currently in a legacy application and 2) a legacy application requires access to data or capabilities in the modernized system.

Requirements:

- Provide coarse grained service API between modernized services and legacy business components and services.
- Encapsulate use case specific logic outside Business Objects.

Solution:

- Implementing a Service Facade between the modernized and legacy components of the applications. The service facade must appear to the modernized services to be a modernized service and to the legacy application like part of the legacy application.

  Even if both the legacy and modernized systems use POJOs or EJBs etc., there can be no shared context between objects on the SOA and legacy side of the Service Facade. While loose coupling and high cohesion is strongly recommended between SOA Business Sub-Layer services, it is an absolute requirement that services be very loosely coupled across the Service Facade. Further, there can be no business logic or procedural logic in the Service Facade, only the logic necessary to map objects from the SOA services and to the legacy systems.

The Service Facade will be located in the Data Access Sub-Layer of the SOA since in essence the Service Facade will be look like a data service interface – with the possibility of some "side effects" from the movement of data across the Service Facade.

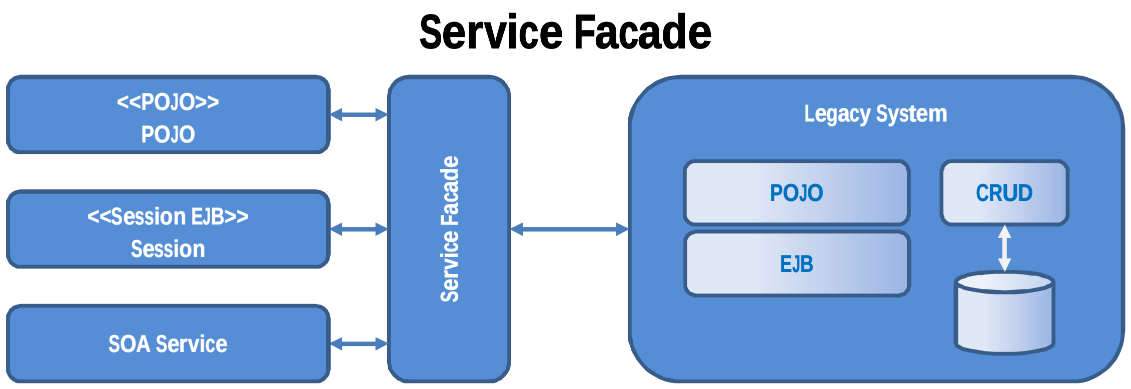


Figure 16 - Service Façade Pattern

Advantages:

As can be seen in the class diagram; designing interfaces shields the core service logic from changes making them more manageable while keeping the consumers un-impacted. As with majority of design patterns this design pattern supports with the principles of loose coupling, increasing cohesion, and enhancing scalability.

3.3.2. **Business Object**

Problem:

There is a need for an architectural pattern to model business domains and map entities within those domains to business entities providing easy access to business functions and associated data.

Requirements:

Separate and centralize business state and related behavior from the remainder of the application. Existing business domains consists of structured interrelated composite objects. Achieve reusability and cohesion of business logic.

Solution:

Business Objects is an object-oriented architectural style focused on modeling a business domain. Business Objects store data values and expose them through properties; contain and manage business data used by the application; and provide stateful programmatic access to the business data and related functionality. Business entities also validate the data contained within the entity and encapsulate business logic to ensure consistency and to implement business rules and behavior. Moreover Business Objects can be implemented using either POJOs or composite objects.

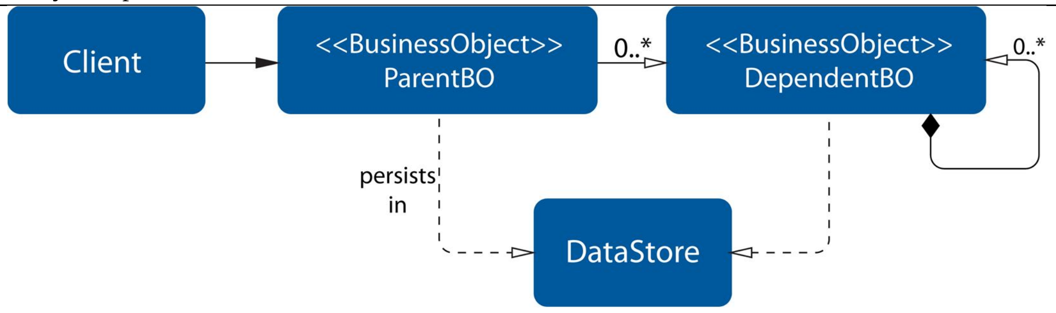


Figure 17 - Business Object Pattern

Advnatges:

Business logic is centralized maximizing reusability. Using this design pattern the principle of "Reusability" and "Cohesion" from Business Sub-Layer architectural considerations is achieved.

3.4. Recommended Best Practices

Table 3 - Recommended Standards for Business Layer

Functionality	Recommended Standard
Message Interfaces	JMS
Service Interfaces	WebServices(SOAP/REST), JAX-WS
Business Interfaces	POJO, EJB3.0 Session Beans
Business Process Management	BPMN 2.0, BPEL
Transaction Management	Container Managed Transactions(CMT), JTA

- Asynchronous messaging communication should be used via Java Message Service (JMS) interfaces or Web Services based on WS-* standards directly or using these protocols over direct vendor specific message queuing products.
- Consider using Synchronous Web Service calls in place of simulated REQUEST/RESPONSE Asynchronous message constructs.
- Design for coarse grained business Sub-Layer access to business services to limit round trips between architectural layers and sub-layers.
- Prefer Statelessness
- Leverage container/framework capabilities for cross cutting concerns like Security, Transaction Management

4. Data Layer / Data Access Sub-Layer

The Data Layer manages access to persistent storage which is usually a relational database but occasionally may be flat files or XML repositories. Persistence can be complex in large applications therefore the Data Access Sub-Layer shields this complexity from the SOA Layer. The data services in the Data Layer communicate with the data services in the Data Access Sub-Layer of the SOA Layer. Making it easier to switch data sources and share data access logic between multiple applications when they share the same data source.

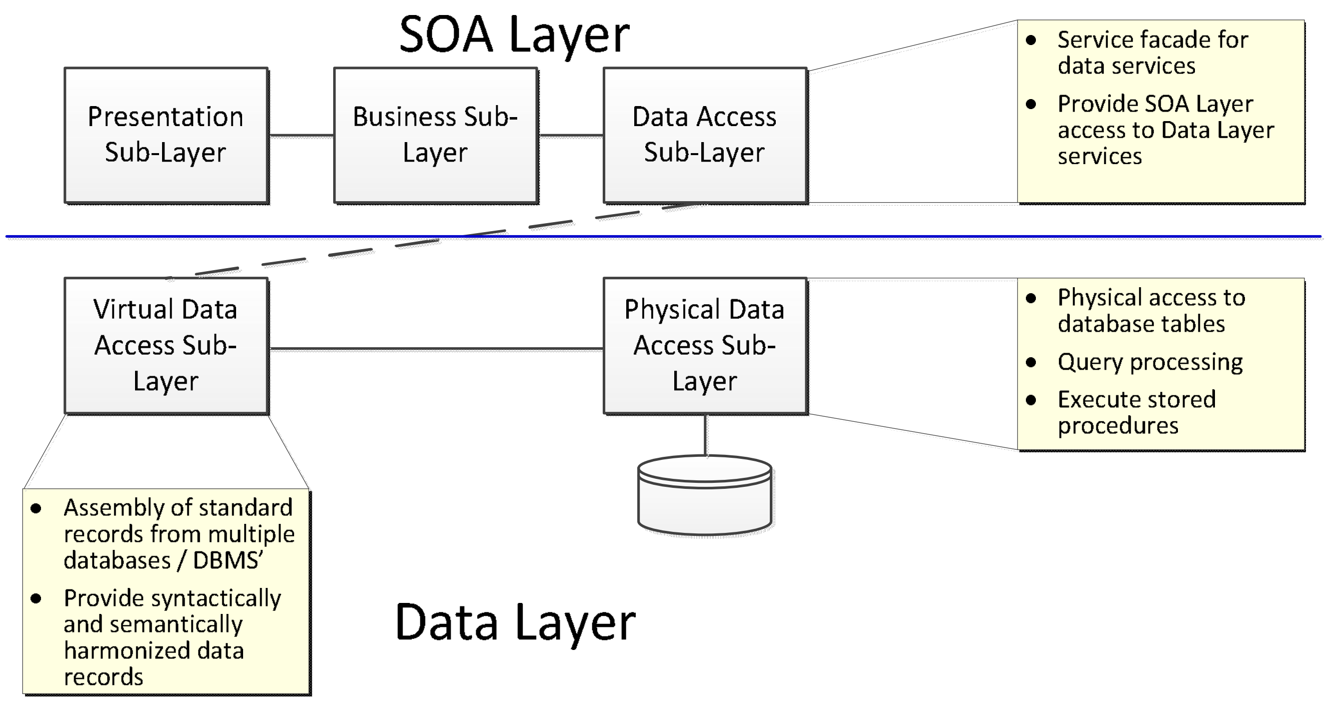

SOA Layer
Presentation Sub-Layer
Business Sub-Layer
Data Access Sub-Layer
Service facade for data services
Provide SOA Layer access to Data Layer services
Virtual Data Access Sub-Layer
Physical Data Access Sub-Layer
Physical access to database tables
Query processing
Execute stored procedures
Assembly of standard records from multiple databases / DBMS'
Provide syntactically and semantically harmonized data records
Data Layer

Figure 18 – Description of EAA Data Services for General Communication between Applications

4.1. Architectural Considerations

Following section describes the general and specific architectural components for the Data Layer.

- **Data Persistence:** Data services in the Data Access Sub-Layer of the SOA Layer understand application constructs such as Entity Beans etc. but does not understand constructs that require knowledge of the DBMS being used, the names of the databases being referenced, or other database specific information such as table names etc. These quantities are confined to the data Layer.
- **Data Security:** Although other layers of the architecture provide a number of security mechanism and all data access requests should be well formed and from a known, authenticated source, this cannot be assumed to be true and the Data Layer will also need to use the DBMS capabilities to assure database security. At the database level security guards not just against malicious attacks, but against accidental damage to the database. For example, the DBMS will need to enforce referential integrity and protect against accidental deletion or modifications of data – actions which while not malicious in intent, but which can wreak havoc on the database.
- **Data validation:** Must provide an effective data validation technique which will validate all input data from other layers or third party components. Must handle NULL values properly, filter out invalid characters and examine user inputs used in dynamic SQL queries for SQL injection attacks. The Data Layer must return informative error messages if validation fails.

- **Queries:** Queries are the primary source of data access in the Data Layer and therefore SQL queries must be optimized to maximize database performance and throughput. Improper or poorly written queries can result in serious performance degradation. Specific design and performance considerations must be taken into account while designing queries.
- **Connection & Transaction management:** The Data Layer depends on the SOA Business Sub-Layer for both connection and transaction management.
- **Performance:** Must consider performance impacts by analyzing the cost of frequently used queries, batching similar queries to reduce round trips to the database while reducing network traffic and must avoid long running batch commands that will lock database resources.
- **Exception Handling:** Must handle exceptions associated with data sources, timeouts and CRUD (Create, Read, Update, and Delete) operations. Depending upon the gravity of the error the Data Layer may pass some exceptions to Business Sub-Layer i.e. for concurrency violations, unresponsive queries, deadlock etc.

4.2. Data Access Layer Components

This section describes the specific guidelines and design patterns to achieve architectural goals for Data Access Sub-Layer.

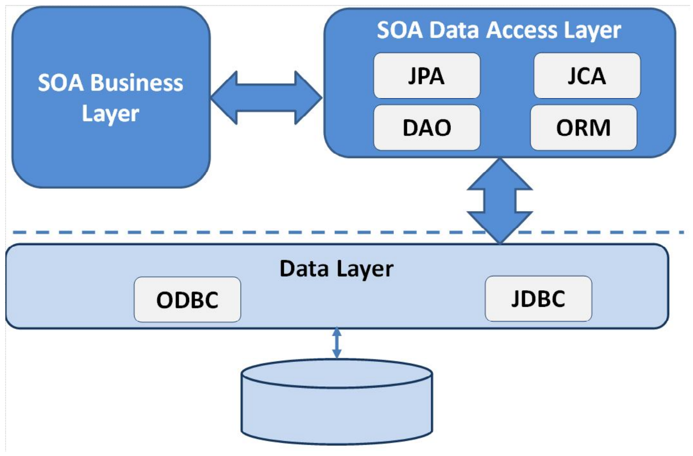


Figure 19 - Data Access Layer Components

4.2.1. Object Relational Mapping

Problem: Application needs to access persistent storage using entity beans which are too heavyweight and tightly coupled with the container. Moreover deployment and testing is also challenging. Need a standardized lightweight persistence methodology with minimal configuration to access data storage.

Requirements:

- Lightweight persistence API for Object Relational Mapping (ORM)
- Decoupling data access responsibilities from ORM.
- Standardizing data access into a lightweight layer and make business components transparent to data access.

Solution:

Use Java Persistence API (JPA) which simplifies entity persistence model by providing a cleaner easier object relational-mapping. JPA eliminates lengthy deployment descriptors through annotations, adds support for named static and dynamic queries, provides JPQL and enhanced EJB QL and provides the ability to be used/test outside the container.

Advantages:

Simplified access to persistent storage is obtained by using the lightweight object-relational mapping persistence API. JPA also results in fewer classes and interfaces and standardizes access which promotes portability and loose coupling with the container itself.

4.2.2. **Data Access Object (DAO)**

Problem:

Applications need to access different sets of persistent storage which are only accessible by different APIs. Moreover business components use of these APIs, which may be scattered across the application, introduces tight coupling with the data source. There is a need to decouple data access logic and encapsulating it into a data access Sub-Layer.

Requirements:

- Need to abstract and decouple implementation of data access.
- Need for JPA access to incompatible legacy data sources.
- Need to make business components transparent to the persistence mechanism.
- Need to facilitate native SQL access
- Need to centralize and optimize complex queries

Solution:

Use a Data Access Object (DAO) to abstract and encapsulate access to the persistent store. The DAO manages the connection with the data source to retrieve and store data. DAO implements the required access mechanism and the business logic can use the simple interfaces exposed by the DAO. The DAO ensures that the interface exposed to the clients does not change even if the underlying data store changes essentially encapsulating the data access logic and acting as an adapter between clients and the data source.

Advantages:

DAOs shield business objects from data access intricacies and make an application more portable by reducing complexity of the business objects and centralizing data access into a separate sub-layer. The downside is that it does add an extra layer or may not be useful in container managed persistence; but the upsides from the separation of business logic and data logic and localizing access to the physical data stores far outweigh the downsides.

4.3. Data Layer Components

4.3.1. **Native SQL Access**

Problem:

Data access through the Physical Sub-Layer features native SQL access which must be written to meet specific performance and / or security considerations.

Requirements:

SQL queries need to maximize database performance, prevent SQL injection attacks, and must be structured to use the DBMS' inherent optimization capabilities.

Solution: The following guidance must be followed in developing SQL queries

a. Must use host variables instead of hard-coded literals in SQL strings which allows the DBMS to use its own optimization and reduces SQL injection attacks.

Bad Practice:

"SELECT USER_NAME FROM USER_INFO WHERE USER_ID = "+ USERID;

Recommended Practice:

"SELECT USER_NAME FROM USER_INFO WHERE USER_ID =?"

b. Must use *PreparedStatement* instead of *Statement* to maximize database optimization.

c. Must use *finally* block to close database variables like *Statement* or *PreparedStatement* etc. Most DBMS' continue to allocate resources when databases are not closed after use. Closing these variables reduces the time and resources spent by the DBMS and ensure that these resources are freed.

d. Must limit use of column functions in *select* lists and they must be kept out of out of *where* the clause.

Advantages:

Allows database to use *index* whereas use of column functions in a *where* clause prohibits the database from using *index* on the column. Moreover, column functions are faster and easier to do in code than if the database is required to do them. Limiting column functions and keeping column functions out of the *where* clause also makes code more portable since not all column functions are supported by all DBMS'.

Bad Practice:

"SELECT USER_NAME FROM USER_INFO WHERE TO_CHAR (HIRE_DT, 'YYYY-MM-DD') >=?

Recommended Practice:

"SELECT USER_NAME FROM USER_INFO WHERE HIRE_DT>=?

e. Must explicitly specify columns list in both *select* and *insert* clauses.

Advantages: If database table is reordered, no code changes are needed.

Bad Practice:

*SELECT * FROM USER_INFO;*

INSERT INTO USER_INFO VALUES ('JOHN', 'DOE', '111 MAIN ST', 'NOWHERE', 'VA');

Recommended Practice:

SELECT USER_NAME, LAST_NAME FROM USER_INFO;

INSERT INTO USER_INFO (USER_NAME, LAST_NAME, STREET_ADDRESS, CITY, STATE) VALUES (?,?,?,?,?);

4.3.2. **Caching**

Data caching can improve application and database performance by limiting the number of remote invocations in distributed applications by reducing the number of database calls to the persistent data stores. Even though caching improves performance it can complicate design and introduce additional complexity by introducing concerns such as concurrent code and cluster-wide synchronization.

Considerations for the selection of a caching framework include;

1. **Integration with the ORM framework:** It should be easy to integrate the caching product with some of the popular ORM products. The domain objects are POJOs and map to RDBMS entities and cached in memory, thereby reducing network traffic to the DBMS.
2. **Persistent Storage:** In case the memory capacity is full, the cache product should evict objects to a local disk.
3. **Support for distributed cache:** A cache within each JVM needs to be coordinated in a clustered environment.
4. **Sharing of objects with a JVM:** All the application threads within a JVM should be able to access the same instance of an object in a cache.
5. **Support for cache invalidation:** The caching product should provide a facility to invalidate a cache or a cache group. Cache invalidation should be coordinated in a distributed cache.
6. **Availability:** The cache maintains a local copy; some operations can continue even if the original source is unavailable.
7. **Scalability:** In a distributed cache, multiple copies of a cache are available across processes. Thus, the scalability of the server is improved.

4.3.3. **Stored Procedures**

Problem:

Application needs to execute high-performance queries and perform batch data manipulation operations.

Requirements:

- Query processing close to DBMS
- Better query optimization
- Ability to execute a set of SQL statements in single call

Solution:

Stored procedures perform intermediate processing on the database server, without transmitting unnecessary data across the network to the client. Only the records that are actually required by the client application are transmitted. Using a stored procedure can result in reduced network usage and better overall performance.

Applications that execute SQL statements one at a time typically cause data to cross the network twice for each SQL statement, once sending the SQL statement to the DBMS and then once to return the results to the client that had requested the data. Unfortunately, much of the data that is returned by the SQL statement may not ultimately be included in the final response and thus, the time and resources required to send the data from the DBMS server to the application server is wasted. A stored procedure can group SQL statements together, making it necessary to only cross the network twice for each group of SQL statements as well as reducing the number of irrelevant records that are returned. Reducing network usage and the length of database locks improves processing time and reduces lock contention problems.

Applications that process large amounts of SQL-generated data, but present only a subset of the data to the user, can generate excessive network usage because all of the data is returned to the client before final processing. A stored procedure can do the processing on the DBMS server, and transmit only the required data to the client.

Advantages:

- Reduced network usage between clients and servers.
- Enhanced hardware and software capabilities
- Improved security

4.4. Recommended Best Practices

Table 4 - Data Access Layer best practices

Functionality	Recommended Standard
CRUD Operations	JPA
High Performance queries	Stored Procedures
Batch Data Manipulation	Stored Procedure
Ad hoc Querying	DAO with Native SQL access
Legacy Data Access	JCA

- Stored Procedures must not contain business logic data.
- Stored Procedures should leverage ORM data validators to ensure data integrity.
- Stored Procedures should use lazy loading as the default association-fetching strategy. However, the ORM context needs to be properly managed to avoid lazy initialization exceptions.
- Stored Procedures must use container managed connection pools to gain maximum performance benefit.